Universidad de Cádiz
Escuela Superior de Ingeniería

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

GENERACIÓN DE UN MÓDULO DE VISUALIZACIÓN PARA EL SIMULADOR DE VUELO PIPER SENECA

AUTOR: SERGIO RUIZ PINO

Cádiz, Mayo 2021

Universidad de Cádiz
Escuela Superior de Ingeniería

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

GENERACIÓN DE UN MÓDULO DE VISUALIZACIÓN PARA EL SIMULADOR DE VUELO PIPER SENECA

DIRECTOR: ALBERTO GABRIEL SALGUERO HIDALGO
CODIRECTOR: LUIS GARCÍA BARRACHINA
AUTOR: SERGIO RUIZ PINO

Cádiz, Mayo 2021

Agradecimientos

A mis tutores, Alberto Salguero y Luis García, por su ayuda, sus sugerencias de mejora y tiempo dedicado.

A mi Madre, por haberme apoyado siempre.

A Alvaro Baro, por tantos años de apoyo y amistad.

A Raquel Recio, por hacer que cada día sea un motivo para seguir adelante.

Resumen

Los simuladores de vuelo son una herramienta esencial para el entrenamiento de pilotos, uno de sus componentes esenciales, para que cumpla su función de forma efectiva, es el generador de imagen. El simulador de vuelo Piper Seneca, formado por cinco equipos conectados mediante una red, pertenecía a la escuela de pilotos de Jerez y fue donado a la universidad de Cádiz. La funcionalidad del simulador es adecuada, pero su generador de imágenes, es decir su módulo visual, que es uno de los equipos que conforman el simulador estaba obsoleto además de ser privativo. El objetivo de este TFG es actualizar dicho módulo. Diseñar un nuevo generador de imágenes de cero totalmente nuevo, adaptarlo a las mejoras de software y hardware que se han producido en los últimos años, además de tener acceso al código del nuevo generador de imágenes para poder seguir mejorándolo a futuro y poder prescindir del módulo antiguo.

Debido al completo desconocimiento de casi todos los aspectos del simulador y al no disponer de documentación de ningún tipo respecto al desarrollo de esté, no sabíamos nada respecto de como realizaba la comunicación con los equipos que lo conforman. El simulador ni siquiera arrancaba si no encontraba el módulo de visualización antiguo en su red. Tras realizar un estudio en profundidad del código del simulador aplicando diversos tipos de técnicas y mediante un proceso de ingeniaría inversa, se ha realiza un correcto protocolo de comunicaciones y se ha conseguido adaptar a nuestro nuevo generador de imágenes, pudiendo así retirar el módulo antiguo y quitar ese dependencia que existe entre el antiguo módulo y el simulador.

Este TFG atravesó siete etapas, se explicará continuación una muy breve descripción del trabajo realizado en cada etapa:

En la primera etapa se realizo un estudio para determinar que herramientas utilizar para desarrollar, para ello se tuvo en cuenta diferentes aspectos tales como licencias, experiencias anteriores y potencia del software que se iba usar. En la segunda etapa se estudio todo lo relacionado con las herramientas elegidas para ganar un nivel de experiencia aceptable y así poder realizar un software aceptable a la altura de las herramientas usadas.

La tercera etapa se caracterizo por el estudio del simulador y su forma de comunicarse, fue necesario la creación de pequeños programas para poder entender de forma aproximada las comunicaciones que realizaba el simulador, ya que no se tenia documentación de nada respecto al desarrollo del simulador. En la cuarta etapa de gran duración, se desarrollo el protocolo de comunicaciones, se destaca la gran dificultad que supuso desarrollar este protocolo de forma correcta debido a todos los aspecto que debíamos tener en cuenta además no se tenia nada con lo que empezar ya que no teníamos ninguna documentación.

En la quinta etapa se comprobó de forma exhaustiva que el protocolo de comunicación era correcto. En la sexta etapa tras comprobar que las comunicaciones eran correctas se termino de desarrollar el módulo de visualización añadiendo los gráficos nuevos y aspectos climatológicos. En la séptima y ultima etapa se hicieron pruebas de software para comprobar que el funcionamiento era el correcto y se termino de elaborar esta memoria.

Abstract

Flight simulators are an essential tool for the training of pilots, one of its essential components, to fulfill its function effectively, is the image generator. The Piper Seneca flight simulator, made up of five computers connected by a network, belonged to the Jerez pilot school and was donated to the University of Cádiz. The simulator's functionality is adequate, but its image generator, that is, its visual module, which is one of the equipment that makes up the simulator, was obsolete as well as being proprietary. The objective of this TFG is to update said module. Design a completely new image generator from scratch, adapt it to the software and hardware improvements that have occurred in recent years, in addition to having access to the code of the new image generator to be able to continue improving it in the future and to be able to do without the module ancient.

Due to the complete ignorance of almost all aspects of the simulator and not having documentation of any kind regarding its development, we did not know anything about how it communicated with the teams that comprise it. The simulator wouldn't even start if it couldn't find the old display module on your network. After conducting an in-depth study of the simulator code applying various types of techniques and through a reverse engineering process, a correct communications protocol has been carried out and it has been adapted to our new image generator, thus being able to remove the old module and remove that dependency that exists between the old module and the simulator.

This TFG went through seven stages, a very brief description of the work carried out in each stage will be explained below:

In the first stage, a study was carried out to determine what tools to use to develop, for this, different aspects such as licenses, previous experiences and power of the software to be used were taken into account. In the second stage, everything related to the chosen tools was studied to gain an acceptable level of experience and thus be able to make an acceptable software at the level of the tools used.

The third stage was characterized by the study of the simulator and its way of communicating, it was necessary to create small programs to be able to roughly understand the communications carried out by the simulator, since there was no documentation of anything regarding the development of the simulator. In the fourth stage of great duration, the communications protocol was developed, the great difficulty of developing this protocol correctly stands out due to all the aspects that we had to take into account, and there was nothing to start with since we did not we had no documentation.

In the fifth stage, it was thoroughly verified that the communication protocol was correct. In the sixth stage, after verifying that the communications were correct, the display module was developed by adding the new graphics and weather aspects. In the seventh and final stage, software tests were carried out to verify that the operation was correct and this report was completed.

Índice general

Índice de figuras

Índice de cuadros

,

Parte I

Prolegómeno

Capítulo 1

Introducción

1.1. Motivación

Este proyecto está motivado por la necesidad de reemplazar el módulo de visualización del simulador de vuelo Piper Seneca, cuya última versión data del inicio de la época del 2000, por lo que este módulo se puede considerar obsoleto, ya que desde el año del lanzamiento se han dado muchísimos avances a nivel de software y hardware que permiten realizar una mejora considerable.

El modulo de visualización es el encargado de generar los gráficos del simulador de vuelo, es decir simularía lo que el piloto vería durante un vuelo, es el encargado de cargar terrenos, cielo, condiciones climatológicas, en general cualquier cosa que sea visible.(véase la figura 1.1)

Figura 1.1: Visualización de Microsoft Flight Simulator.

El simulador de vuelo es una gran herramienta que no podíamos permitir que quedará en el olvido, ya que el módulo antiguo además de estar obsoleto, tenía muchísimas limitaciones y no podríamos hacer modificaciones debido a que no disponemos de su código fuente. Gracias a este reemplazo, se dispondrá de una gran mejora tanto a nivel de usuario como de desarrollador.

1.2. Alcance

Este proyecto consiste en la realización de un programa que permita conectarse con el simulador de vuelo Piper Seneca y recree el entorno gráfico, es decir lo que el piloto vería durante un ensayo. El objetivo no es desarrollar de forma completa el simulador de vuelo sino desarrollar un nuevo módulo de visualización para el simulador aprovechando que otros aspecto de la simulación son controlados desde otros programas.

El antiguo módulo de visualización estaba anticuado y además era privativo, con el nuevo módulo se ha conseguido que el simulador no quede obsoleto además de tener el código del

nuevo software y poder seguir ampliando sus funcionalidades.

1.3. Glosario de términos

En este apartado se describen algunos términos que son necesarios conocer antes de comenzar a leer esta memoria:

- **Piper Seneca** Nombre del simulador de vuelo.
- **Dynamics** Es el programa principal del simulador de vuelo y desde el que se realizan todos los cálculos.
- **IOS** Programa perteneciente al simulador que controla el inicio de la simulación y condiciones climáticas.
- **Screens** Programa perteneciente al simulador que controla las pantallas de la cabina.
- **Hardware** Programa del simulador que controla que todo el hardware del simulador funciona correctamente.
- **Visual** Programa de visualización del simulador que pretendemos sustituir.
- **VisualPiperSeneca** El nuevo módulo de visualización que se ha desarrollado en este proyecto.
- **C++** Lenguaje de programación orientado a objetos.
- **Unreal Engine 4** Motor de diseño de videojuegos diseñado por Epic Games.

1.4. Estructura del documento

El documento se encuentra dividido en nueve bloques:

- Simuladores de vuelo.
- Planificación.
- Requisitos del sistema.
- Análisis del sistema.
- Diseño del sistema.
- Implementación del sistema
- Pruebas de software.
- Conclusión.
- Manuales.

Capítulo 2

Simuladores de vuelo

En este capítulo se resumirá un poco la historia de los simuladores de vuelo y presentaremos el simulador de vuelo Piper Seneca, que se encuentra en la escuela superior de ingeniería de la universidad de Cádiz.

2.1. Historia de los simuladores de vuelo

La simulación de vuelo es una industria que mueve miles de millones de euros en todo el mundo, y el origen de esta disciplina se remonta al origen mismo de la aviación. Allá por principios del siglo XX, los primeros pilotos de aviones con motor se entrenaban usando aviones reales, con distintas potencias y en distintas fases que iban desde simplemente ser pasajero, hasta poco a poco ir tomando control de aviones más cercanos a los que después tendrían que pilotar. Más tarde, se fue viendo la necesidad de algún tipo de aparato que permitiese a los alumnos aprender sin poner en peligro su vida, y así fue como aparecieron los primeros dispositivos de simulación de las características de vuelo. Estos primeros "simuladores", consistían en aparatos construidos con partes de aviones reales anclados al suelo, que usando el viento y respondían a las fuerzas aerodinámicas de éste para habituar así a los pilotos, aunque debido a la dependencia del viento, estos aparatos no tuvieron mucho éxito. (véase figura 2.1)

Figura 2.1: Simulador que usaba el viento.

El enorme aumento en la necesidad de entrenar a muchos más pilotos durante la primera guerra mundial, dio lugar a muchos avances en la simulación, entre ellos la introducción de test psicológicos y de medición con aparatos electrónicos de los tiempos de reacción ante los distintos estímulos a los que los individuos evaluados eran expuestos. De esta forma, se

seleccionaba a los pilotos.

Nuevos avances permitieron sustituir a operadores humanos, por aparatos mecánicos o eléctricos que, siendo controlados por un entrenador, simulaban características del vuelo como turbulencias por viento o altitud, cambios en la inclinación o rachas de viento debidas a la velocidad.

El simulador de este tipo, que mayor éxito tuvo fue el "Link Trainer" (véase figura 2.2), patentado en 1930, desarrollado por Edwin Link, el cual es considerado el padre de la simulación. Tanto este como los otros aparatos de este tipo, se calibraban mediante prueba y error hasta que el diseñador "sentía" que el aparato estaba bien ajustado.

A pesar de todos estos avances, estos aparatos de entrenamiento de vuelo no tuvieron mucha acogida y no se consideraban un buen sustituto de los vuelos para entrenar a los pilotos.

A partir de 1930 otra línea de aparatos fue siendo desarrollada: los simuladores de instrumentos de vuelo. Edwin Link y su escuela de vuelo diseñaron varios de estos simuladores, los cuales tuvieron mucho más éxito y fueron vendidos en muchos países, hasta que en 1937 American Airlines fue la primera aerolínea del mundo en comprar un aparato Link Trainer para entrenar a sus pilotos. Estos aparatos simulaban diversos instrumentos de vuelo, y permitían distintas funcionalidades, como son los trazadores de curso, grabar la posición y el recorrido del simulador y simular la comunicación del transmisor del avión con una señal de radio permitiendo al alumno comunicarse con el instructor. Estos avances, junto con más avances electrónicos, permitieron a la simulación ganar mucho más reconocimiento para el entrenamiento de pilotos. En estos años podemos encontrar los primeros simuladores con sistemas visuales. Estos sistemas visuales utilizaban un bucle de película que simulaba los efectos de movimiento de cabeceo y balanceo.

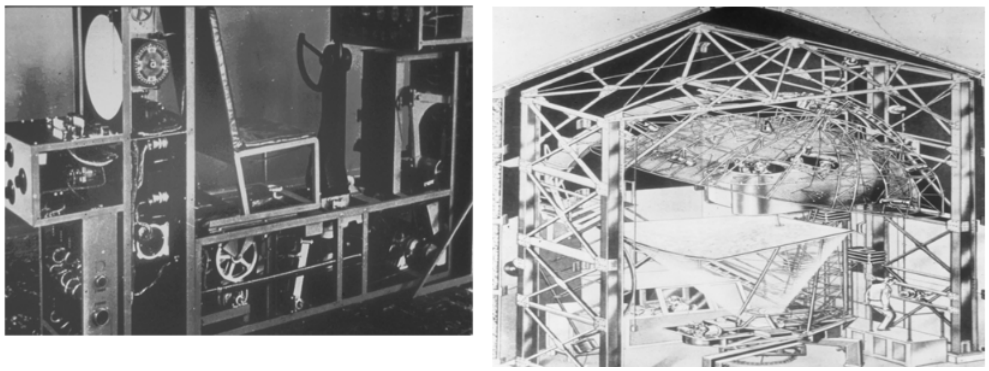

Figura 2.2: Simulador LinkTrainer.

La segunda guerra mundial dio lugar a otro enorme incremento en la necesidad de entrenar a un gran número de pilotos. Durante este periodo, los aparatos de entrenamiento incluyeron muchas nuevas características que simulaban diversas características del vuelo real.

Destacable de este periodo, es el desarrollo en 1939-1941, de un enorme aparato que simulaba los movimientos de las estrellas en el cielo nocturno, para permitir a los pilotos entrenar la ubicación utilizando las estrellas, simulando así las características de los aviones que tenían que cruzar el océano Atlántico durante la segunda guerra mundial. Distintos aparatos, para entrenar habilidades concretas como la que acabamos de mencionar, se desarrollaron durante esta época.

El desarrollo de ordenadores analógicos, permitió simular respuestas a las fuerzas aerodinámicas en lugar de simplemente duplicarlas de forma empírica. Estos son los primeros ancestros de los simuladores de vuelo modernos.

Todos estos avances eliminaron cualquier duda acerca de la utilidad y la efectividad de los simuladores de vuelo para entrenar a nuevos pilotos. Esto impulsó el desarrollo de innumerables aparatos. Entre ellos tenemos el desarrollo de un simulador que emulaba por completo el funcionamiento de un Boeing 377 Stratocruiser, el cual fue el primer simulador de vuelo completo en ser propiedad de una aerolínea. Simuladores como este, y otros similares desarrollados durante las décadas de los 40 y 50, utilizaban la computación análoga, que si bien permitía mucha mayor precisión que aparatos previos, tenía el inconveniente de que con el aumento en la cantidad de procesamiento necesaria, traía un número de errores mucho mayor a la cantidad de precisión que se conseguía con ese aumento en la capacidad de procesamiento.

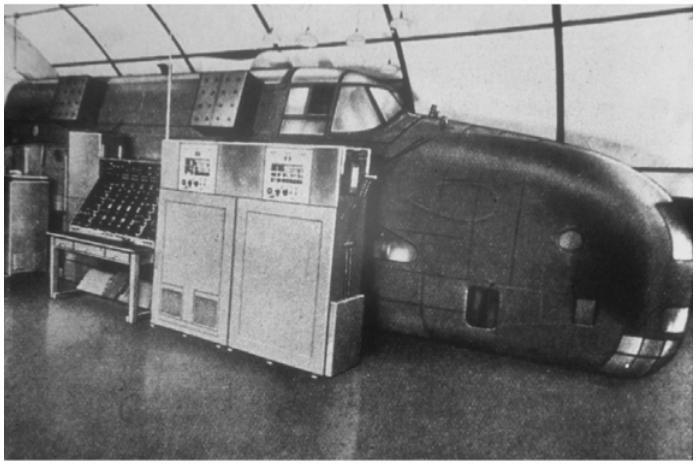

Figura 2.3: Simulador que usaba computación analógica.

Se hizo obvio que se había llegado al límite de la computación análoga, y así fue como los simuladores digitales fue ganando popularidad, hasta que a principios de los años 60, la compañía Link desarrolló su primer simulador digital en tiempo real, el Mark 1, el cual fue sin lugar a dudas el avance de más éxito.

Uno de los mayores avances en la simulación de vuelo, tanto para ganar credibilidad y aceptación, como para permitir su uso y desarrollo de forma generalizada en muchos países y por distintas empresas de forma conjunta, ha sido el desarrollo de standards comunes. Esto, por una parte, permitió a las autoridades regular las características y la calidad de los simuladores para asegurarse unos niveles de calidad mínimos que fuesen seguros para los pilotos. Y por otra parte, estos estándares permitieron a las compañías estandarizar sus diseños de forma generalizada para que todos los pilotos recibiesen el mismo entrenamiento. [1]

2.2. Simulador de vuelo Piper Seneca

Piper Seneca es el simulador de vuelo que se encuentra en el sótano de la Escuela Superior de Ingeniaría de Cádiz (véase figura 2.4), es un modelo profesional de entrenamiento de pilotos y el elemento principal de este proyecto ya que sin el simulador este proyecto no existiría.

Figura 2.4: Simulador de vuelo Piper Seneca.

El simulador está formado por cinco ordenadores conectados a un rack que forman una red local en estrella(vease figura 2.5), donde un equipo es el servidor (Dynamics) y los demás equipos son clientes (véase figura 2.6).

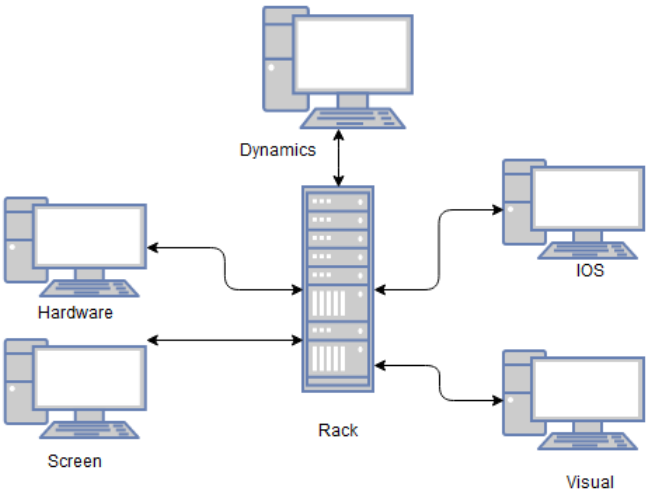

Figura 2.5: Diagrama de red del simulador.

Figura 2.6: Rack del simulador.

2.2.1. Dynamics

Es el cerebro del simulador, es el encargado de realizar todos los cálculos numéricos y llevarlos a los demás equipos, actúa de servidor (véase figura 2.7).

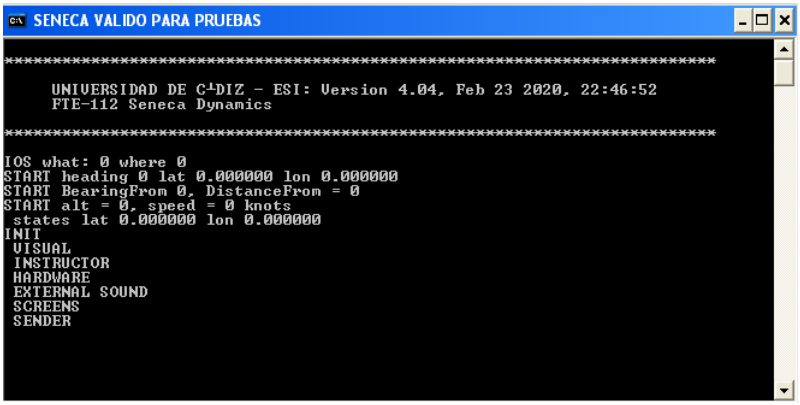

Figura 2.7: Programa Dynamics.

2.2.2. IOS

Es el equipo encargado de controlar la simulación. Desde este equipo se gestiona el inicio de la simulación, posición de inicio de la simulación, condiciones climatológicas, condiciones de vuelo tales como fallos de motor, nivel de combustible y otros aspectos (véase figura 2.8).

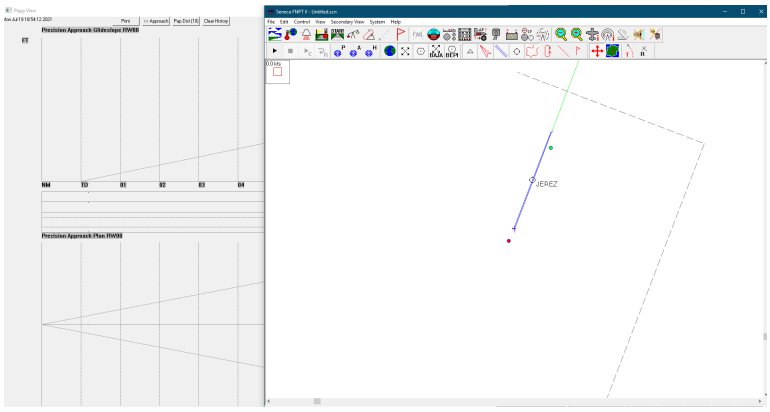

Figura 2.8: Programa IOS.

2.2.3. Hardware

El simulador está compuesto por varias tarjetas que controlan aspectos como la fuerza de los mandos, movimientos de los mandos y botonera de la cabina. El equipo hardware es encargado de comprobar que todas las tarjetas están conectadas y además que realizan su función de forma correcta sin fallos.

Las tarjetas están alojadas en el morro del simulador en un rack (véase figura 2.9).

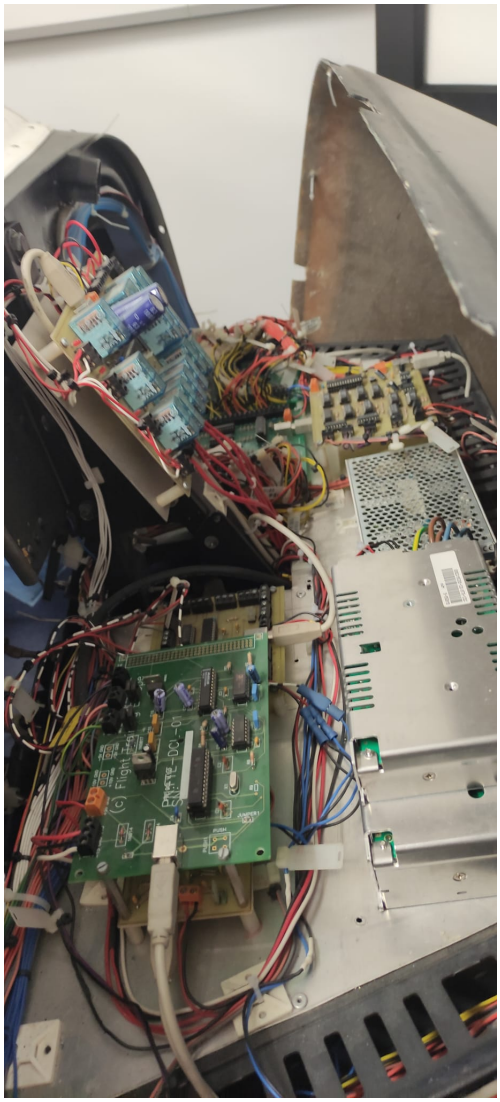

Figura 2.9: Tarjetas internas del simulador.

2.2.4. Visual

Este equipo es el encargado de gestionar la visualización de la simulación, es el módulo a sustituir (véase figura 2.10).

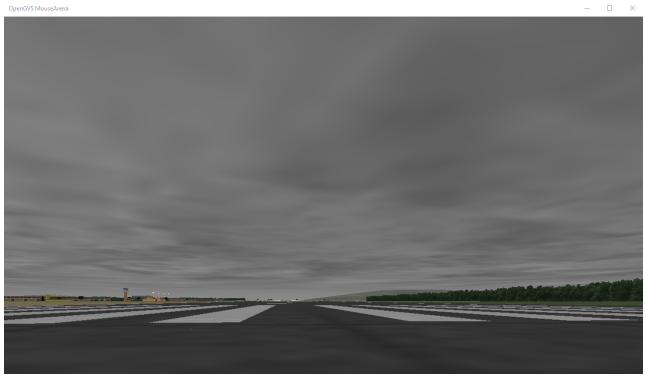

Figura 2.10: Visual antiguo.

2.2.5. Screens

Equipo encargado de mostrar las imágenes de las pantallas que se encuentran en la cabina del simulador (véase figura 2.11).

Figura 2.11: Cabina del simulador de vuelo.

Capítulo 3

Planificación

En este capitulo se indicará la planificación seguida, los costes del proyecto y las herramientas utilizadas para su desarrollo.

3.1. Etapas

El modelo de desarrollo de software usado para el desarrollo de nuestro software ha sido el modelo en cascada. Según los criterios dados por Gómez y Moraleda [2], el modelo en cascada consiste en dividir en distintas etapas el desarrollo del software, cada fase se desarrolla de forma independiente y no podremos comenzar una fase si todas las fase previas no están terminadas(véase figura 3.1).

Es por ello que se ha decidido utilizar este tipo de modelo ya que por las necesidades de desarrollo, el proyecto debía pasar por varias etapas y cada etapa era un prerrequisito de la siguiente. Cada etapa ha estado marcada por un conjunto de objetivos y una duración en la que estos objetivos tenían que estar completados, también han surgido distintos problemas que se han abordado para poder continuar a la siguiente etapa.

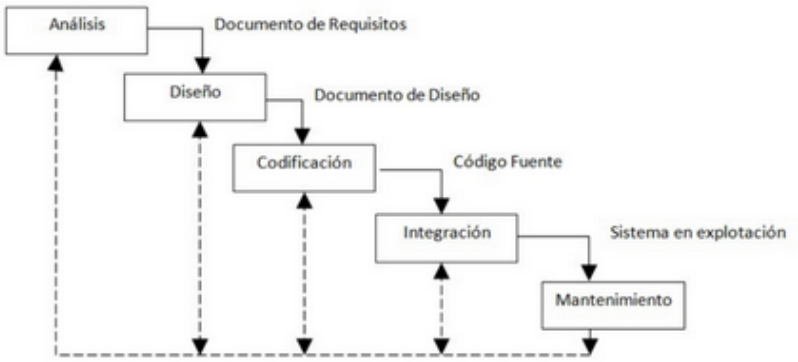

Figura 3.1: Ciclo de desarrollo en cascada.

3.1.1. Etapa de inicio

La primera etapa se caracterizó por la realización de un estudió para determinar cuales eran los requisitos del proyecto, que herramientas y lenguajes de programación serían los más óptimos para desarrollar una solución, además de realizar una estimación del tiempo

necesario para poder terminar el proyecto, esta etapa coincide con las etapas de análisis y de diseño del ciclo en cascada.

3.1.2. Etapa de aprendizaje

En esta esta etapa, el objetivo principal era familiarizarse y obtener cierta destreza con el motor Unreal Engine 4, ya que no se tenían conocimientos previos y debido a la magnitud del problema era necesario obtener un gran conocimiento.

Se realizaron diversas prueba básica con UE4 usando diferentes técnicas de desarrollo para obtener una destreza media del motor.

3.1.3. Etapa de estudio del código fuente del simulador

Durante esta etapa, se estudio todo el código fuente del programa Dynamics, ya que este equipo actúa como servidor y antes de comenzar era necesario saber como funcionaban las comunicaciones con los otros equipos para poder sustituir Visual.

3.1.4. Etapa de desarrollo de comunicación

El objetivo en esta etapa fue entender y adaptar las comunicaciones que realizaba el simulador de vuelo con el módulo antiguo, debido a la complejidad de como se realizaban las comunicaciones en ambas plataformas de desarrollo, esta etapa tuvo una gran duración ya que se tuvo que estudiar de forma detenida el envío de datos en ambas parte y encontrar como adaptar de forma correcta la recepción de datos para que pudieran ser recibidos por ambas plataformas.

3.1.5. Etapa de interacción con el sistema

El objetivo principal del proyecto se desarrolló en esta etapa, ya que tras recoger todas las comunicaciones e interacciones con el simulador, en ambos sentidos, se tuvo que adaptar las respuestas de ambas aplicaciones para que la comunicación fuera correcta.

La complejidad para que todo se adaptara y funcionará de forma correcta fue muy elevada y requirió de mucho trabajo, tiempo y recursos. Se utilizaba un conjunto de códigos y estructura para las comunicaciones que se tuvieron que adaptar uno a uno y luego comprobar que era correcta la adaptación.

En esta etapa también se incluyo los aspectos visuales nuevos y las condiciones climáticas.

3.1.6. Etapa de prueba y de creación de documentos

Etapa final de proyecto, en esta etapa se realizaron diversas pruebas de diferentes tipos para comprobar que el software desarrollado funcionaba de forma correcta y también en esta etapa comenzó la creación de documento y manuales.

3.2. Diagrama de Gantt

Tal y como dice Rodriguez [3] 'Un diagrama de Gannt, es una representación gráfica y simultánea de planificación de procesos', así pues se ha diseñado un diagrama de Gantt con el tiempo empleado en cada etapa para tener una visión gráfica del tiempo empleado, se adjuntan a continuación dos diagramas de Gantt(véase figuras 3.3, 3.2).

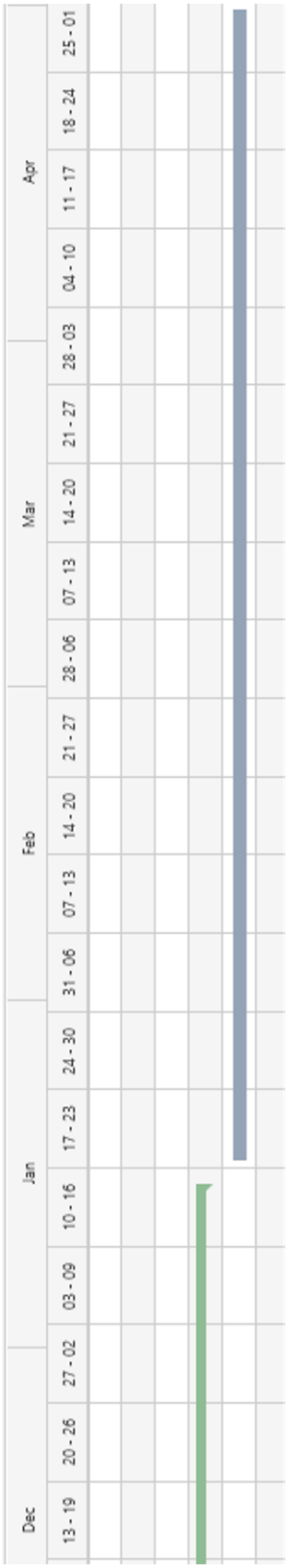

Figura 3.2: Diagrama Gantt 2

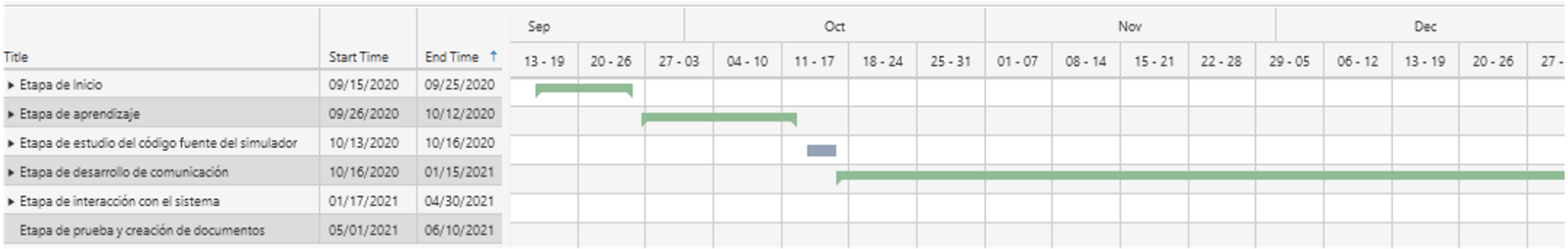

Figura 3.3: Diagrama Gantt 1

3.3. Costes

Un proyecto de estas magnitudes tiene unos costes asociados que son necesarios asumir para poder llegar al objetivo, estos costes pueden ser humanos, materiales o asociados a servicios. Se describen a continuación los costes que se han tenido en la elaboración de este proyecto durante los diez meses de desarrollo.

3.3.1. Costes Humanos

Para estimar el coste humano, debemos de tener en cuenta el número de personas que han desarrollado el proyecto (1) y el nivel de experiencia para poder estimar el salario mensual, en este caso la experiencia es de nivel novel y por tanto se ha estimado un salario de 1450€. En la estimación de salario se ha tenido en cuenta el salario neto tras pagar los impuestos necesarios.

Conceptos	Precio	Meses	Total
Salario	1450€	10	14500€
TOTAL			14500€

Cuadro 3.1: Cuadro de costes humanos

3.3.2. Costes Materiales

El material usado ha sido el ordenador en el que se ha desarrollado el proyecto, para estimar un coste se tendrá en cuenta la vida media del equipo y calcularemos el gasto de uso durante la duración del proyecto, para ello dividiremos entre los meses de vida media y el resultado de esta operación lo multiplicaremos por el número de meses de desarrollo.

Conceptos	Precio	Meses vida media	Meses	Total/año
Precio	900€	84	10	107€
TOTAL				107€

Cuadro 3.2: Cuadro de coste material

3.3.3. Costes asociados a servicios

En este apartados se incluyen los costes asociados a servicios, como son el acceso a internet y el uso de electricidad, como la duración del proyecto ha sido de diez meses, debemos de tener en cuento los gastos causados durante este intervalo de tiempo.

Conceptos	Precio	Meses	Total
Internet/mes	60€	10	600€
Electricidad/mes	45€	10	450€
TOTAL			1050€

Cuadro 3.3: Cuadro de gastos asociados a servicios

3.3.4. Costes totales

Calculados los costes anteriores, el coste total es la suma de todos los costes:

Conceptos	Precio
Coste Humano	14500€
Coste Material	107€
Coste Servicio	1050€
TOTAL	15657€

Cuadro 3.4: Cuadro de gastos totales

3.4. Herramientas utilizadas

En este apartado se indicarán las herramientas usadas en el proyecto.

3.4.1. Herramientas de desarrollo

Las herramientas usadas para el desarrollo de VisualPiperSeneca han sido el motor de videojuegos Unreal Engine 4 y el IDE Microsoft Visual Studio:

Unreal Engine 4 (Véase figura 3.4) Es un motor de videojuegos creado por Epic Games, se ha elegido este motor debido a que es uno de los motores más potentes a la hora de crear contenidos, además de su extendido uso en el mercado. Se ha usado la versión 4.26.

Figura 3.4: Icono Unreal Engine 4.

Microsoft Visual Studio (Véase figura 3.5) Es un entorno de desarrollo integrado que permite el desarrollo de código en múltiples lenguajes. Es el entorno más adecuado para trabajar junto a Unreal Engine 4 debido a la integración entre ambos. Se ha usado la versión de 2019.
.

Figura 3.5: Icono Microsoft Visual Studio

GitHub (Véase figura 3.6) Es una plataforma que permite subir proyecto y llevar un control de versiones utilizando el sistema Git, se adjunta el link del repositorio donde se encuentra el código desarrollado.

`https://github.com/SergioRuizPino/VisualPiperSeneca`.

Figura 3.6: Icono de GitHub

3.4.2. Herramientas de redacción de texto

Para la redacción de esta memoria se ha usado el editor de texto LaTeX(Véase figura 3.7) desde la web OverLeaf. LaTeX permite la generación de textos científicos de gran calidad de redacción y es por ello que se ha elegido esta herramienta, además de las facilidades al poder importar los texto al formato de archivos PDF.

Figura 3.7: Icono LaTeX

3.4.3. Herramientas alternativas

En este apartado se explicarán herramientas alternativas que se estudiaron para ser utilizadas en el desarrollo pero que finalmente se decidió no utilizar.

Unity (Véase figura 3.8) Es un motor de videojuegos creado por Unity Technologies. Aunque su uso es más sencillo que el de Unreal, se decidió no usar este motor debido a que no es tan potente como Unreal, además de tener asociados costes de licencia.

Figura 3.8: Icono de Unity

Microsoft Word (Véase figura 3.9) Es un procesador de textos de uso muy sencillo, debido a costes de licencia y el resultado final de los textos, se decidió no usar.

Figura 3.9: Icono de Microsoft Word

Parte II

Desarrollo

Capítulo 4

Requisitos del sistema

En este capítulo se indicarán los requisitos y objetivos del proyecto.

4.1. Requisitos

Según dice Piattini [4] un requisito es ' Una condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado'.

Sabemos que todos los sistemas tienen requisitos y tras un análisis de estos, se indicarán los requisitos del sistema diseñado para este proyecto.

4.1.1. Requisitos funcionales

Los requisitos funcionales según nos dice García y García [5] tienen la siguiente definición, los requisitos funcionales describen la funcionalidad o los servicios que se espera que el sistema tenga y son afirmaciones sobre los servicios que el sistema debe ofrecer.

Tras conocer la definición de lo que son los requisitos funcionales, pasaremos a continuación a detallar los requisitos funcionales del sistema:

REF-01	Permitir iniciar el sistema.
Descripción	El sistema debe ser capaz de iniciar.

Cuadro 4.1: Requisito Funcional 01

REF-02	Permitir salir del sistema.
Descripción	El sistema debe ser capaz de salir.

Cuadro 4.2: Requisito Funcional 02

REF-03	Permitir conectarse con Dynamics.
Descripción	El sistema debe ser capaz de establecer una conexión.

Cuadro 4.3: Requisito Funcional 03

REF-05	Permitir enviar datos mediante una conexión.
Descripción	El sistema debe ser capaz de enviar datos mediante una conexión creada previamente.

Cuadro 4.5: Requisito Funcional 05

REF-04	Permitir recibir datos mediante una conexión.
Descripción	El sistema debe ser capaz de recibir datos mediante una conexión creada previamente.

Cuadro 4.4: Requisito Funcional 04

REF-06	Movimiento en el eje x.
Descripción	El sistema debe ser capaz de mover el elemento avión en el eje x.

Cuadro 4.6: Requisito Funcional 06

REF-07	Movimiento en el eje y.
Descripción	El sistema debe ser capaz de mover el elemento avión en el eje y.

Cuadro 4.7: Requisito Funcional 07

REF-08	Movimiento en el eje z.
Descripción	El sistema debe ser capaz de mover el elemento avión en el eje z.

Cuadro 4.8: Requisito Funcional 08

REF-9	Rotación en el eje x.
Descripción	El sistema debe ser capaz de rotar el elemento avión en el eje x.

Cuadro 4.9: Requisito Funcional 09

REF-10	Rotación en el eje y.
Descripción	El sistema debe ser capaz de rotar el elemento avión en el eje y.

Cuadro 4.10: Requisito Funcional 10

REF-11	Rotación en el eje z.
Descripción	El sistema debe ser capaz de rotar el elemento avión en el eje z.

Cuadro 4.11: Requisito Funcional 11

REF-12	Lectura de fichero .XML.
Descripción	El sistema debe leer e interpretar los datos de ficheros de extensión XML.

Cuadro 4.12: Requisito Funcional 12

REF-13	Lectura de fichero .RAW.
Descripción	El sistema debe ser capaz de leer e interpretar los datos de ficheros de extensión RAW.

Cuadro 4.13: Requisito Funcional 13

REF-14	Creación de terrenos.
Descripción	El sistema debe ser capaz de crear terrenos según los datos de ficheros XML Y RAW.

Cuadro 4.14: Requisito Funcional 14

REF-15	Inicio un vuelo.
Descripción	El sistema debe ser capaz de iniciar un vuelo.

Cuadro 4.15: Requisito Funcional 15

REF-16	Pausar un vuelo.
Descripción	El sistema debe ser capaz de pausar un vuelo.

Cuadro 4.16: Requisito Funcional 16

REF-17	Detener un vuelo.
Descripción	El sistema debe ser capaz de detener un vuelo.

Cuadro 4.17: Requisito Funcional 17

REF-18	Condiciones climáticas: Nubes.
Descripción	El sistema debe capaz de recrear las condiciones climáticas de las nubes.

Cuadro 4.18: Requisito Funcional 18

REF-19	Condiciones climáticas: Niebla.
Descripción	El sistema debe ser capaz de recrear las condiciones climáticas de niebla.

Cuadro 4.19: Requisito Funcional 19

REF-20	Ciclo día noche.
Descripción	El sistema debe ser capaz de recrear el ciclo día-noche.

Cuadro 4.20: Requisito Funcional 20

REF-21	Detección de colisión.
Descripción	El sistema debe ser capaz de detectar una colisión.

Cuadro 4.21: Requisito Funcional 21

REF-22	Posición de inicio.
Descripción	El sistema debe ser capaz leer la posición de inicio del objeto avión.

Cuadro 4.22: Requisito Funcional 22

4.1.2. Requisitos no funcionales

Según Sommerville [6], "Los requisitos no funcionales son los que no se refieren directamente a una función específica del sistema sino a las propiedades emergentes del propio sistema", así pues, en este apartado indicaremos cuales son los requisitos no funcionales del sistema.

RENF-01	Seguridad.
Descripción	El sistema no precisa de seguridad, debido a que la información que maneja no es sensible.

Cuadro 4.23: Requisito No Funcional 01

RENF-02	Usabilidad.
Descripción	El sistema debe ser amigable e intuitivo a la hora de su uso.

Cuadro 4.24: Requisito No Funcional 02

RENF-03	Compatibilidad.
Descripción	El sistema debe ser compatible con la información que recibe y con los ficheros que procesa.

Cuadro 4.25: Requisito No Funcional 03

RENF-04	Portabilidad.
Descripción	El sistema esta diseñado para funcionar bajo el sistema operativo Windows, pero como se entrega el código fuente, podría portarse en un futuro a otras plataformas.

Cuadro 4.26: Requisito No Funcional 04

4.2. Objetivos

En esta sección se indican los objetivos del proyecto:

OBT-01	Vuelo.
Descripción	El sistema debe ser capaz de permitir la recreación total (Inicio a Fin) de un vuelo usando la cabina de vuelo.

Cuadro 4.27: Objetivo 01

OBT-02	Terrenos Nuevos.
Descripción	El sistema debe ser capaz de permitir la creación de terrenos nuevos que posteriormente aparecerán en los vuelo.

Cuadro 4.28: Objetivo 02

Capítulo 5

Análisis del sistema

En este capítulo se describe el análisis realizado a VisualPiperSeneca desde el punto de vista de la ingeniería de software.

5.1. Modelo de casos de Uso

En este apartado describiremos los casos de uso del sistema, según Vega [7] los casos de uso describen la interacción de los usuarios con el sistema.

5.1.1. Caso de uso Inicio del sistema

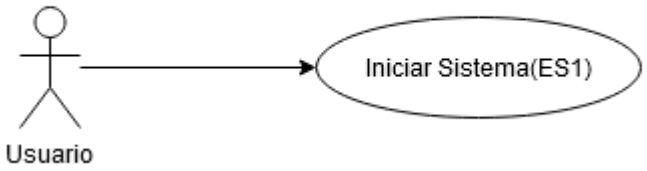

Figura 5.1: Caso de uso Inicio del Sistema.

Descripción del escenario ES1

- **CU 1** Iniciar Sistema
- **Descripción** EL usuario iniciar el equipo para su uso (REF-01).
- **Precondición** Ninguna.
- Pasos
  1. El usuario inicia los equipos implicados en el simulador.
  2. El usuario inicia el programa Dynamics en el equipo correspondiente.
  3. El usuario inicia el programa IOS en el equipo correspondiente.
  4. El usuario inicia el programa Hardware en el equipo correspondiente.
  5. El usuario inicia el programa Screens en el equipo correspondiente.
  6. El usuario debe posicionarse en la carpeta donde reside VisualPiperSeneca

- **Postcondición** El usuario inicia el sistema.

5.1.2. Casos de uso tras el inicio del sistema

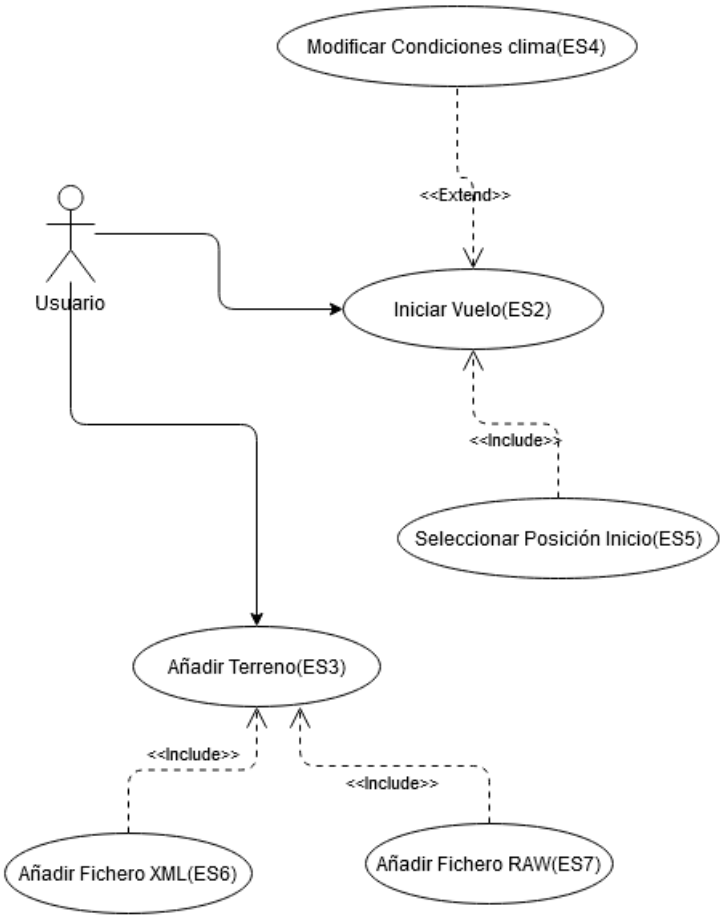

Figura 5.2: Caso de uso tras el Inicio del Sistema.

Descripción del escenario ES2

- **CU 2** Iniciar vuelo.
- **Descripción** El usuario desea inicia un vuelo (REF-01,REF-15,REF-03,REF-04,REF-05)
- **Precondición** EL usuario debe iniciar el equipo(ES1).
- Pasos
  1. El usuario inicia VisualPiperSeneca
  2. El usuario selecciona en IOS la posición de comienzo (include Seleccionar posición) (ES5)

3. El usuario pulsa el botón play en IOS.
4. Si el usuario desea, cambie las condiciones climáticas pulsando nubes (extend Modificar posiciones clima) (ES4).
5. El usuario se posiciona en la cabina de vuelo y pulsa el botón de pausa del simulador.

- **Postcondición** El usuario inicia un vuelo.

Descripción del escenario ES3

- **CU 3** Añadir Terreno.
- **Descripción** El usuario desea añadir un terreno (REF-12,REF-13,REF-14)
- **Precondición** Ninguna.
- Pasos
  1. Añadir Ficheros XML (include Añadir Fichero XML)(ES6).
  2. Añadir Fichero RAW (include Añadir Fichero RAW)(ES7).
- **Postcondición** El terreno aparece en la visualización.

Descripción del escenario ES4

- **CU 4** Modificar Condiciones clima.
- **Descripción** El usuario desea modificar el clima en la visualización (REF-18,REF-19,REF-20).
- **Precondición** El usuario debe modificar el clima durante el inicio del vuelo (ES2).
- Pasos
  1. El usuario indica el nivel de nubes.
  2. El usuario indica el nivel de niebla.
  3. El usuario indica la hora de comienzo.
  4. El usuario pulsa aceptar.
- **Postcondición** En la visualización se mostrarán cambios respecto a las condiciones climáticas.

Descripción del escenario ES5

- **CU 5** Seleccionar Posición de inicio.
- **Descripción** El usuario indica la posición de comienzo del vuelo (REF-18,REF-19,REF-20).
- **Precondición** El usuario indica la posición de inicio durante inicio del vuelo (ES2).
- Pasos
  1. El usuario indica la altura.
  2. El usuario indica la posición.
  3. El usuario pulsa aceptar.

- **Postcondición** El usuario comenzará el vuelo en la posición elegida.

Descripción del escenario ES6

- **CU 6** Añadir Fichero XML.
- **Descripción** El usuario añade un fichero XML para crear un terreno(REF-12).
- **Precondición** El fichero existe y los datos son correctos.
- Pasos
  1. El usuario renombra el fichero indicando la posición del terreno.
  2. El usuario copia el fichero a la carpeta terrenos ubicada dentro de la carpeta de VisualPiperSeneca.
- **Postcondición** El fichero aparece en la carpeta terrenos.

Descripción del escenario ES7

- **CU 7** Añadir Fichero RAW.
- **Descripción** El usuario añade un fichero RAW para crear un terreno(REF-13).
- **Precondición** El Fichero existe y los datos son correctos.
- Pasos
  1. El usuario renombra el fichero indicando la posición del terreno.
  2. El usuario copia el fichero a la carpeta terrenos ubicada dentro de la carpeta de VisualPiperSeneca.
- **Postcondición** El fichero aparece en la carpeta terrenos.

5.1.3. Casos de uso durante el vuelo

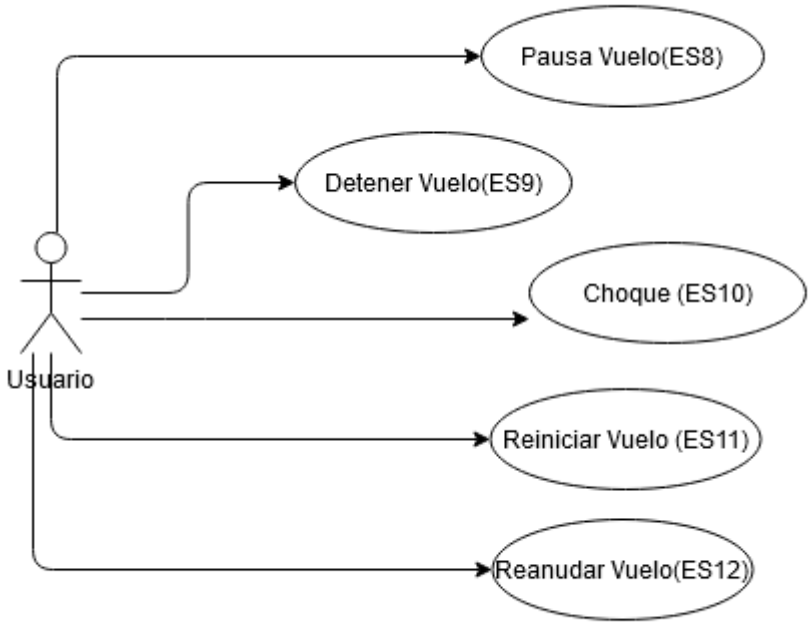

Figura 5.3: Caso de uso durante el vuelo.

Descripción del escenario ES8

- **CU 8** Pausar Vuelo.
- **Descripción** El usuario pausa el vuelo(REF-16).
- **Precondición** El usuario se encuentra realizando un vuelo.
- Pasos
  1. El usuario pulsa el botón de pausa en la cabina de vuelo o el botón pausa desde Hardware.
- **Postcondición** El vuelo entrará en modo pausa.

Descripción del escenario ES9

- **CU 9** Detener Vuelo.
- **Descripción** El usuario detiene el vuelo(REF-17).
- **Precondición** El usuario se encuentra realizando un vuelo.
- Pasos
  1. El usuario pulsa el botón de parada desde IOS.
- **Postcondición** El vuelo se detendrá.

Descripción del escenario ES10

- **CU 10** Choque.
- **Descripción** El usuario choca con el terreno durante el vuelo(REF-21).
- **Precondición** El usuario se encuentra realizando un vuelo.
- Pasos
  1. El usuario choca con un terreno.
  2. El vuelo se detiene.
- **Postcondición** El vuelo se detendrá.

Descripción del escenario ES11

- **CU 11** Reiniciar Vuelo.
- **Descripción** El usuario reinicia un vuelo tras detenerlo o chocar(REF-15).
- **Precondición** El usuario ha detenido el vuelo o ha tenido un choque.
- Pasos
  1. El usuario pulsa el botón play desde IOS si es necesario.
  2. El usuario pulsa el botón pausa de la cabina de vuelo.
- **Postcondición** El vuelo se iniciará.

Descripción del escenario ES12

- **CU 12** Reanudar Vuelo.
- **Descripción** El usuario reanuda un vuelo tras realizar una pausa(REF-15).
- **Precondición** El usuario ha pausado el vuelo (ES8).
- Pasos
  1. El usuario pulsa el botón play desde IOS si es necesario.
  2. El usuario pulsa el botón pausa de la cabina de vuelo.
- **Postcondición** El vuelo se iniciará.

5.1.4. Casos de uso durante el vuelo, movimiento

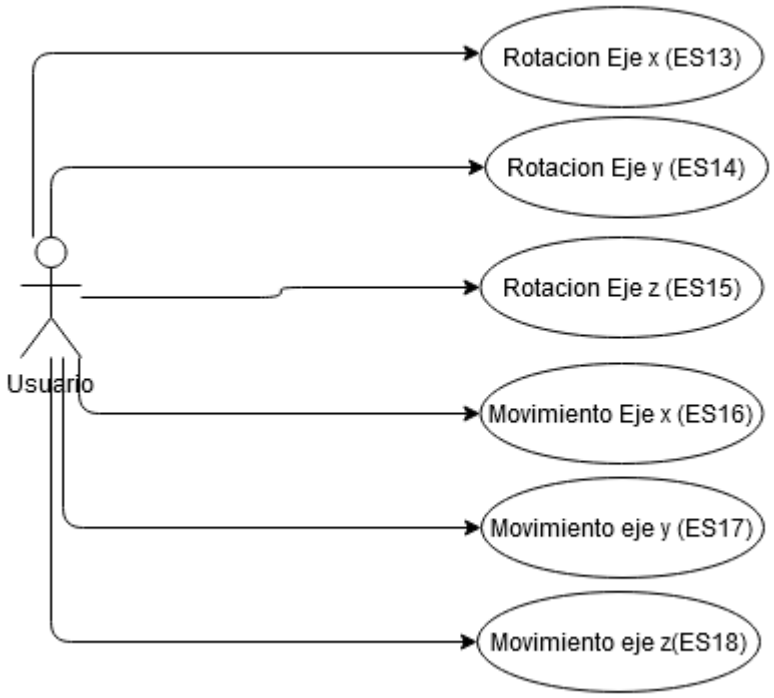

Figura 5.4: Caso de uso durante el vuelo, movimiento.

Descripción del escenario ES13

- **CU 13** Rotación eje x.
- **Descripción** El usuario realiza una rotación desde los mandos del simulador(REF-09).
- **Precondición** El usuario está en un vuelo (ES2).
- Pasos
  1. El usuario mueve el mando de vuelo realizando un giro hacia el eje x.
  2. El avión girará y se mostrará en pantalla.
- **Postcondición** El avión actualizará su posición.

Descripción del escenario ES14

- **CU 14** Rotación eje y.
- **Descripción** El usuario realiza una rotación desde los mandos del simulador(REF-10).
- **Precondición** El usuario está en un vuelo (ES2).
- Pasos
  1. El usuario mueve el mando de vuelo realizando un giro hacia el eje y.

2. El avión girará y se mostrará en pantalla.

- **Postcondición** El avión actualizará su posición.

Descripción del escenario ES15

- **CU 15** Rotación eje z.
- **Descripción** El usuario realiza una rotación desde los mandos del simulador(REF-11).
- **Precondición** El usuario está en un vuelo (ES2).
- Pasos
  1. El usuario mueve el mando de vuelo realizando un giro hacia el eje z.
  2. El avión girará y se mostrará en pantalla.
- **Postcondición** El avión actualizará su posición.

Descripción del escenario ES16

- **CU 16** Movimiento eje x.
- **Descripción** El usuario realiza una movimiento desde los mandos del simulador(REF-06).
- **Precondición** El usuario está en un vuelo (ES2).
- Pasos
  1. El usuario mueve el mando de vuelo realizando un movimiento hacia el eje x.
  2. El avión se moverá y se mostrará en pantalla.
- **Postcondición** El avión actualizará su posición.

Descripción del escenario ES17

- **CU 17** Movimiento eje y.
- **Descripción** El usuario realiza una movimiento desde los mandos del simulador(REF-07).
- **Precondición** El usuario está en un vuelo (ES2).
- Pasos
  1. El usuario mueve el mando de vuelo realizando un movimiento hacia el eje y.
  2. El avión se moverá y se mostrará en pantalla.
- **Postcondición** El avión actualizará su posición.

Descripción del escenario ES18

- **CU 18** Movimiento eje z.
- **Descripción** El usuario realiza una movimiento desde los mandos del simulador(REF-08).
- **Precondición** El usuario está en un vuelo (ES2).

- Pasos
  1. El usuario mueve el mando de vuelo realizando un movimiento hacia el eje z.
  2. El avión se moverá y se mostrará en pantalla.
- **Postcondición** El avión actualizará su posición.

5.2. Modelo de comportamiento

Según Pressman [8], el modelo de comportamiento indica la forma en la que responderá el software a eventos o estímulos externos. Este modelo consta de un diagrama de secuencia en el que se muestra la interacción del sistema con el usuario, además de el contrato de operaciones, que es una pequeña descripción de las operaciones que se realizan.

5.2.1. Caso de uso iniciar sistema.

CU1 Iniciar Sistema

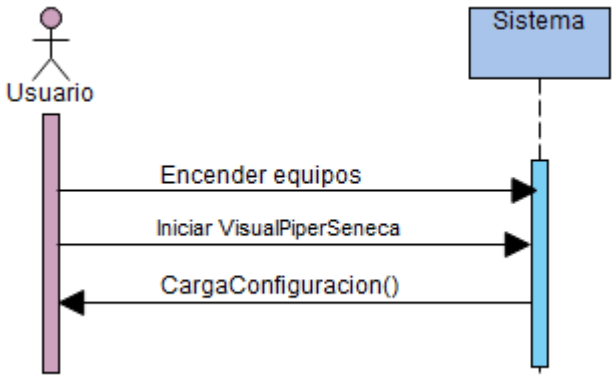

Figura 5.5: Diagrama de secuencia CU1.

- Encender equipos:
  - Actores : Usuario.
  - Responsabilidades : El usuario inicia los equipo y los programas relacionados con el simulador.
  - Precondición : Ninguna.
  - Postcondición : El equipo inicia.
- Iniciar VisualPiperSeneca
  - Actores : Usuario.
  - Responsabilidades : El usuario inicia VisualPiperSeneca.
  - Precondición : El equipo está encendido.
  - Postcondición : VisualPiperSeneca inicia.
- CargaConfiguracion()

- Actores : Sistema.
- Responsabilidades : El sistema carga la configuración.
- Precondición : VisualPiperSeneca está en funcionamiento.
- Postcondición : VisualPiperSeneca carga la configuración y queda en espera.

5.2.2. Casos de uso tras iniciar sistema.

CU2 Iniciar vuelo

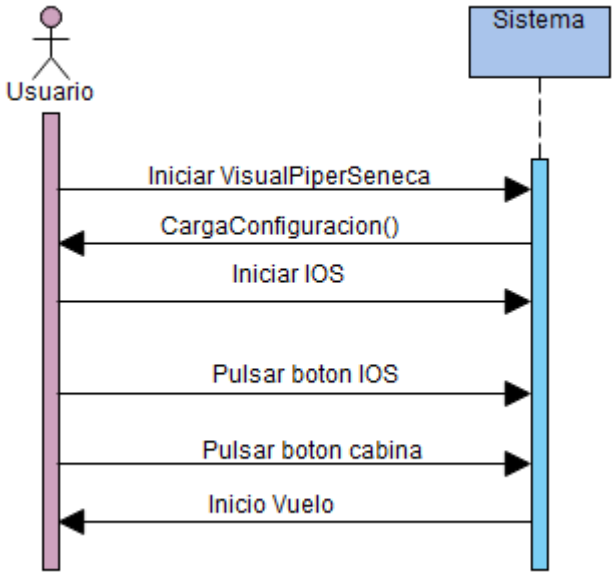

Figura 5.6: Diagrama de secuencia CU2.

- Iniciar VisualPiperSeneca
  - Actores : Usuario.
  - Responsabilidades : El usuario inicia VisualPiperSeneca.
  - Precondición : El equipo está encendido.
  - Postcondición : VisualPiperSeneca inicia.
- CargaConfiguracion()
  - Actores : Sistema.
  - Responsabilidades : El sistema carga la configuración.
  - Precondición : VisualPiperSeneca está en funcionamiento.
  - Postcondición : VisualPiperSeneca carga la configuración y queda en espera.
- Iniciar IOS
  - Actores : Usuario.
  - Responsabilidades : El usuario inicia el programa IOS.
  - Precondición : VisualPiperSeneca está en funcionamiento.

- Postcondición : IOS inicia y queda a la espera.

- Pulsar botón IOS
  - Actores : Usuario.
  - Responsabilidades : El usuario pulsa el botón play en el programa IOS.
  - Precondición : VisualPiperSeneca e IOS están en funcionamiento.
  - Postcondición : El vuelo inicia en modo pausa.
- Pulsar botón cabina
  - Actores : Usuario.
  - Responsabilidades : El usuario pulsa el botón pausa en la cabina de vuelo.
  - Precondición : VisualPiperSeneca e IOS están en funcionamiento y el botón play ha sido pulsado en IOS.
  - Postcondición : El vuelo inicia.

CU3 Añadir Terreno

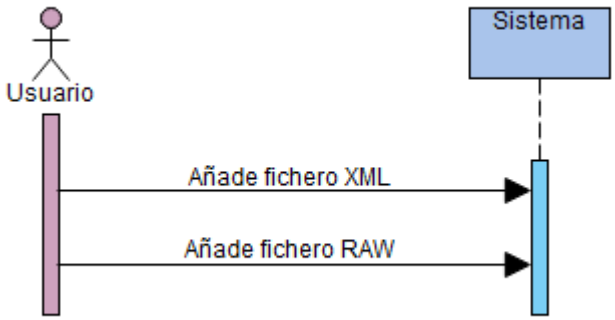

Figura 5.7: Diagrama de secuencia CU3.

- Añadir fichero XML
  - Actores : Usuario.
  - Responsabilidades : El usuario añade un fichero de tipo XML a la carpeta terrenos.
  - Precondición : El equipo está encendido.
  - Postcondición : El fichero queda almacenado en la carpeta terrenos.
- Añadir fichero RAW
  - Actores : Usuario.
  - Responsabilidades : El usuario añade un fichero de tipo RAW a la carpeta terrenos.
  - Precondición : El equipo está encendido.
  - Postcondición : El fichero queda almacenado en la carpeta terrenos.

CU4 Modificar condiciones climáticas

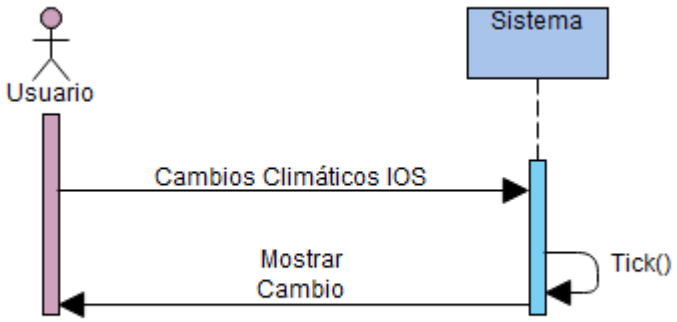

Figura 5.8: Diagrama de secuencia CU4.

- Cambios climáticos IOS
  - Actores : Usuario.
  - Responsabilidades : El usuario modifica en IOS las condiciones climáticas.
  - Precondición : VisualPiperSeneca se está ejecutando durante un vuelo en pausa.
  - Postcondición : VisualPiperSeneca recibe los valores nuevos de las condiciones climáticas.
- Tick()
  - Actores : Sistema.
  - Responsabilidades : El sistema procesa los nuevos datos.
  - Precondición : VisualPiperSeneca ha recibido valores nuevos para las condiciones climáticas.
  - Postcondición : El sistema procesa los nuevos datos para las condiciones climáticas.
- Mostrar Cambio
  - Actores : Sistema.
  - Responsabilidades : El sistema muestra por pantalla el cambio.
  - Precondición : VisualPiperSeneca ha procesado los cambios.
  - Postcondición : El sistema muestra por pantalla los cambios climáticos.

CU5 Seleccionar posición de inicio.

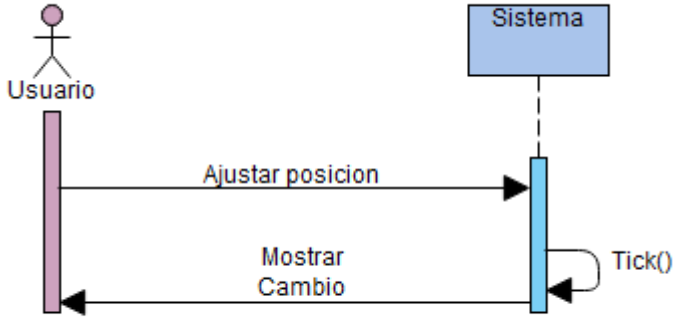

Figura 5.9: Diagrama de secuencia CU5.

- Ajustar posición
  - Actores : Usuario.
  - Responsabilidades : El usuario modifica en IOS la posición inicial de vuelo.
  - Precondición : VisualPiperSeneca se está ejecutando y esta en espera.
  - Postcondición : VisualPiperSeneca recibe los valores de la posición de inicio.
- Tick()
  - Actores : Sistema.
  - Responsabilidades : El sistema procesa los nuevos datos.
  - Precondición : VisualPiperSeneca ha recibido valores nuevos para la posición inicial de vuelo.
  - Postcondición : El sistema procesa los nuevos datos para la posición inicial de vuelo.
- Mostrar Cambio
  - Actores : Sistema.
  - Responsabilidades : El sistema muestra por pantalla el cambio.
  - Precondición : VisualPiperSeneca ha procesado los cambios.
  - Postcondición : El sistema muestra por pantalla la posición nueva de inicio.

CU6 Añadir fichero XML.

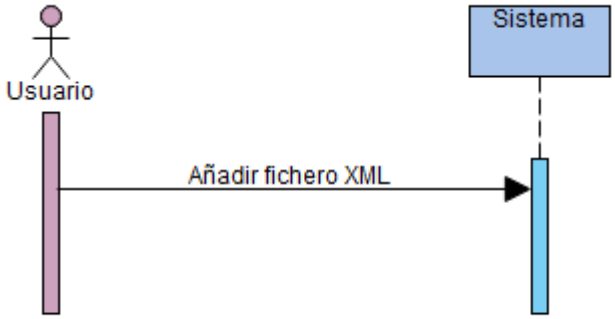

Figura 5.10: Diagrama de secuencia CU6.

- Añadir fichero XML
  - Actores : Usuario.
  - Responsabilidades : El usuario añade un fichero de tipo XML a la carpeta terrenos.
  - Precondición : El equipo está encendido.
  - Postcondición : El fichero queda almacenado en la carpeta terrenos.

CU7 Añadir fichero RAW.

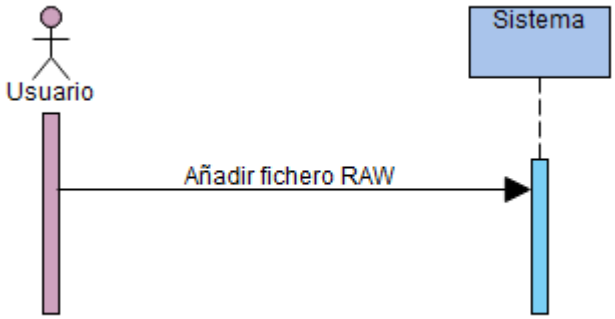

Figura 5.11: Diagrama de secuencia CU7.

- Añadir fichero RAW
  - Actores : Usuario.
  - Responsabilidades : El usuario añade un fichero de tipo RAW a la carpeta terrenos.
  - Precondición : El equipo está encendido.
  - Postcondición : El fichero queda almacenado en la carpeta terrenos.

5.2.3. Casos de uso durante el vuelo

CU8 Pausar vuelo.

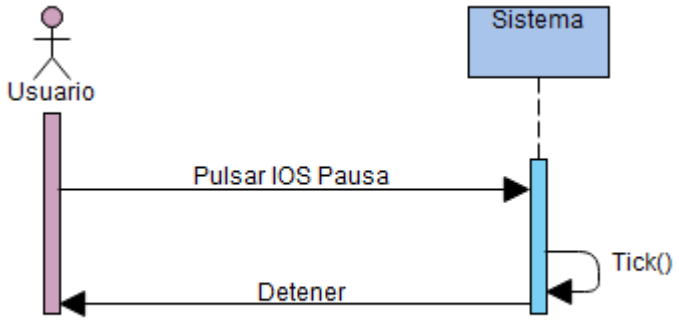

Figura 5.12: Diagrama de secuencia CU8.

- Pulsar IOS Pausa
  - Actores : Usuario.
  - Responsabilidades : El usuario pulsa pausa en hardware o en la cabina de vuelo.
  - Precondición : VisualPiperSeneca se está ejecutando y está en un vuelo.
  - Postcondición : VisualPiperSeneca recibe la orden de pausa.
- Tick()
  - Actores : Sistema.
  - Responsabilidades : El sistema procesa los nuevos datos.
  - Precondición : VisualPiperSeneca ha recibido valores nuevos para el estado de la simulación.
  - Postcondición : El sistema procesa los nuevos datos sobre el estado de simulación.
- Detener
  - Actores : Sistema.
  - Responsabilidades : El sistema detiene la simulación.
  - Precondición : VisualPiperSeneca ha procesado los datos sobre el estado de la simulación.
  - Postcondición : El sistema pausa la simulación y queda en estado de espera.

CU9 Detener vuelo.

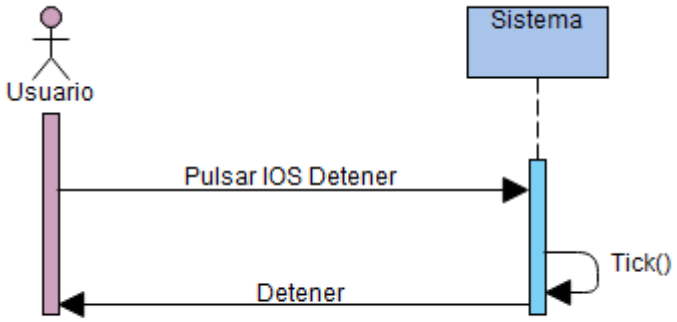

Figura 5.13: Diagrama de secuencia CU9.

- Pulsar IOS Detener
  - Actores : Usuario.
  - Responsabilidades : El usuario pulsa detener en IOS.
  - Precondición : VisualPiperSeneca se está ejecutando y está en un vuelo.
  - Postcondición : VisualPiperSeneca recibe la orden de detener.
- Tick()
  - Actores : Sistema.
  - Responsabilidades : El sistema procesa los nuevos datos.
  - Precondición : VisualPiperSeneca ha recibido valores nuevos para el estado de la simulación.
  - Postcondición : El sistema procesa los nuevos datos sobre el estado de simulación.
- Detener
  - Actores : Sistema.
  - Responsabilidades : El sistema detiene la simulación.
  - Precondición : VisualPiperSeneca ha procesado los datos sobre el estado de la simulación.
  - Postcondición : El sistema detiene la simulación y queda en estado de espera de comenzar un nuevo vuelo.

CU10 Choque.

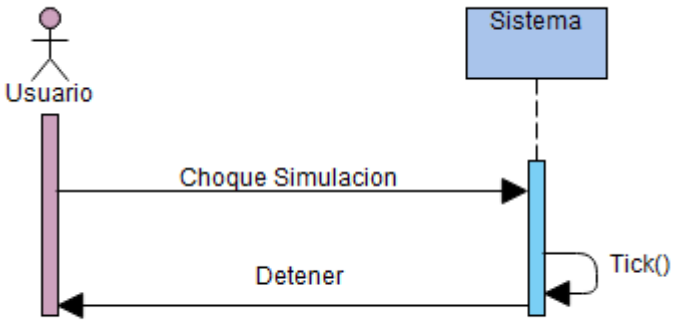

Figura 5.14: Diagrama de secuencia CU10.

- Choque Simulación
  - Actores : Usuario.
  - Responsabilidades : El usuario choca durante una simulación.
  - Precondición : VisualPiperSeneca se está ejecutando y está en un vuelo.
  - Postcondición : VisualPiperSeneca recibe que el usuario a chocado.
- Tick()
  - Actores : Sistema.
  - Responsabilidades : El sistema procesa los nuevos datos.
  - Precondición : VisualPiperSeneca ha recibido valores nuevos para el estado de la simulación.
  - Postcondición : El sistema procesa los nuevos datos sobre el estado de simulación.
- Detener
  - Actores : Sistema.
  - Responsabilidades : El sistema detiene la simulación.
  - Precondición : VisualPiperSeneca ha procesado los datos sobre el estado de la simulación.
  - Postcondición : El sistema detiene la simulación y queda en estado de espera de comenzar un nuevo vuelo.

CU11 Reiniciar Vuelo.

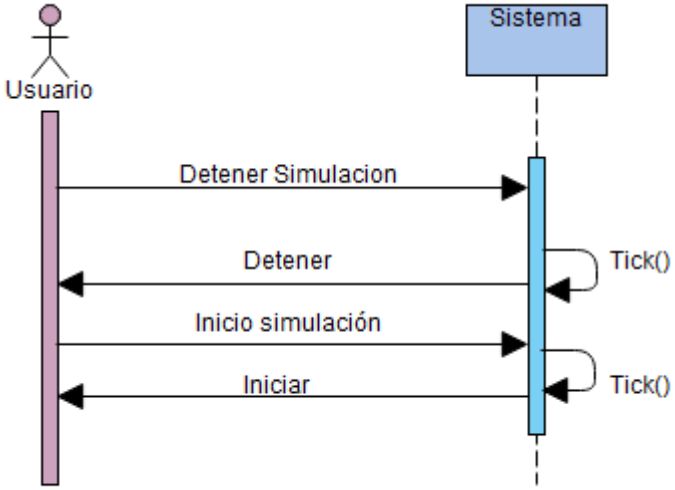

Figura 5.15: Diagrama de secuencia CU11.

- Detener simulación
  - Actores : Usuario.
  - Responsabilidades : El usuario pulsa detener la simulación en IOS.
  - Precondición : VisualPiperSeneca se está ejecutando y esta en un vuelo.
  - Postcondición : El sistema recibe la orden de detener vuelo.
- Tick()
  - Actores : Sistema.
  - Responsabilidades : El sistema procesa los nuevos datos.
  - Precondición : VisualPiperSeneca ha recibido valores nuevos para el estado de la simulación.
  - Postcondición : El sistema procesa los nuevos datos sobre el estado de simulación.
- Detener
  - Actores : Sistema.
  - Responsabilidades : El sistema detiene la simulación.
  - Precondición : VisualPiperSeneca ha procesado los datos sobre el estado de la simulación.
  - Postcondición : El sistema detiene la simulación y queda en estado de espera de comenzar un nuevo vuelo.
- Inicio simulación
  - Actores : Usuario.
  - Responsabilidades : El usuario pulsa play en IOS.
  - Precondición : VisualPiperSeneca se encuentra en estado de espera.
  - Postcondición : El sistema recibe la orden de iniciar vuelo.
- Tick()

- Actores : Sistema.
- Responsabilidades : El sistema procesa los nuevos datos.
- Precondición : VisualPiperSeneca ha recibido valores nuevos para el estado de la simulación.
- Postcondición : El sistema procesa los nuevos datos sobre el estado de simulación.

- Iniciar
  - Actores : Sistema.
  - Responsabilidades : El sistema inicia la simulación.
  - Precondición : VisualPiperSeneca ha procesado los datos sobre el estado de la simulación.
  - Postcondición : El sistema inicia la simulación.

CU12 Reanudar vuelo.

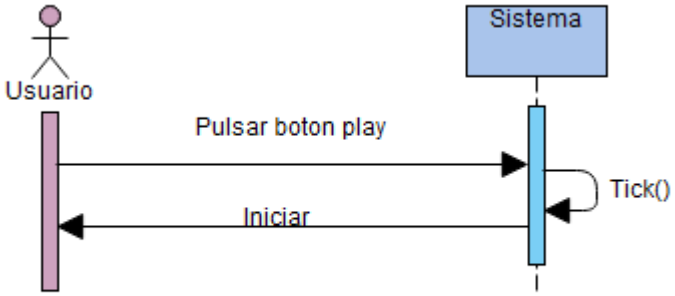

Figura 5.16: Diagrama de secuencia CU12.

- Pulsar botón play
  - Actores : Usuario.
  - Responsabilidades : El usuario pulsa play en IOS.
  - Precondición : VisualPiperSeneca se está ejecutando y está en un vuelo previamente pausado.
  - Postcondición : VisualPiperSeneca recibe la orden de detener.
- Tick()
  - Actores : Sistema.
  - Responsabilidades : El sistema procesa los nuevos datos.
  - Precondición : VisualPiperSeneca ha recibido valores nuevos para el estado de la simulación.
  - Postcondición : El sistema procesa los nuevos datos sobre el estado de simulación.
- Detener
  - Actores : Sistema.
  - Responsabilidades : El sistema inicia la simulación.
  - Precondición : VisualPiperSeneca ha procesado los datos sobre el estado de la simulación.
  - Postcondición : El sistema inicia la simulación.

5.2.4. Casos de uso durante el vuelo, movimiento

CU13 Rotación eje x.

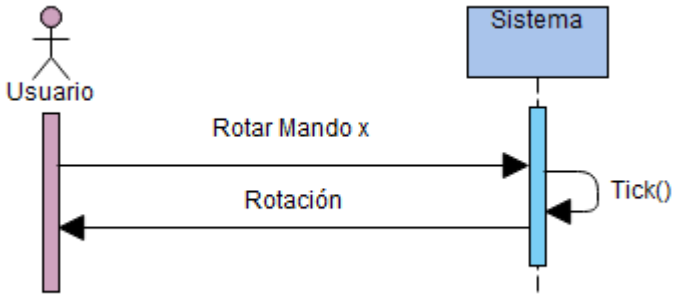

Figura 5.17: Diagrama de secuencia CU13.

- Rotar mando x
  - Actores : Usuario.
  - Responsabilidades : El usuario realiza un movimiento de rotación en el eje x desde los mandos del simulador.
  - Precondición : VisualPiperSeneca se está ejecutando y está en un vuelo.
  - Postcondición : VisualPiperSeneca recibe la orden de giro.
- Tick()
  - Actores : Sistema.
  - Responsabilidades : El sistema procesa los nuevos datos.
  - Precondición : VisualPiperSeneca ha recibido valores nuevos para el estado de la simulación.
  - Postcondición : El sistema procesa los nuevos datos sobre el estado de simulación.
- Rotación
  - Actores : Sistema.
  - Responsabilidades : El sistema procede a realizar una rotación.
  - Precondición : VisualPiperSeneca ha procesado los datos sobre el giro del avión en la simulación.
  - Postcondición : El sistema muestra por pantalla el giro del avión en el eje x.

CU14 Rotación eje y.

Figura 5.18: Diagrama de secuencia CU14.

- Rotar mando y
  - Actores : Usuario.
  - Responsabilidades : El usuario realiza un movimiento de rotación en el eje y desde los mandos del simulador.
  - Precondición : VisualPiperSeneca se está ejecutando y está en un vuelo.
  - Postcondición : VisualPiperSeneca recibe la orden de giro.
- Tick()
  - Actores : Sistema.
  - Responsabilidades : El sistema procesa los nuevos datos.
  - Precondición : VisualPiperSeneca ha recibido valores nuevos para el estado de la simulación.
  - Postcondición : El sistema procesa los nuevos datos sobre el estado de simulación.
- Rotación
  - Actores : Sistema.
  - Responsabilidades : El sistema procede a realizar una rotación.
  - Precondición : VisualPiperSeneca ha procesado los datos sobre el giro del avión en la simulación.
  - Postcondición : El sistema muestra por pantalla el giro del avión en el eje y.

CU15 Rotación eje z.

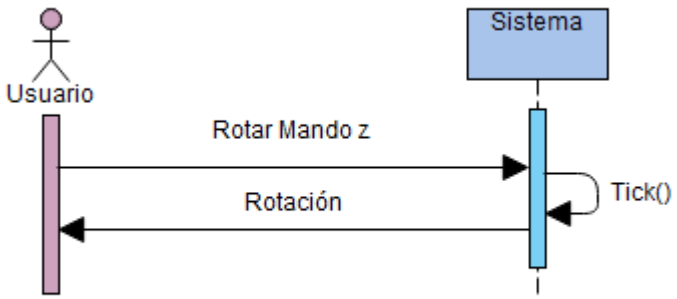

Figura 5.19: Diagrama de secuencia CU15.

- Rotar mando z
  - Actores : Usuario.
  - Responsabilidades : El usuario realiza un movimiento de rotación en el eje z desde los mandos del simulador.
  - Precondición : VisualPiperSeneca se está ejecutando y está en un vuelo.
  - Postcondición : VisualPiperSeneca recibe la orden de giro.
- Tick()
  - Actores : Sistema.
  - Responsabilidades : El sistema procesa los nuevos datos.
  - Precondición : VisualPiperSeneca ha recibido valores nuevos para el estado de la simulación.
  - Postcondición : El sistema procesa los nuevos datos sobre el estado de simulación.
- Rotación
  - Actores : Sistema.
  - Responsabilidades : El sistema procede a realizar una rotación.
  - Precondición : VisualPiperSeneca ha procesado los datos sobre el giro del avión en la simulación.
  - Postcondición : El sistema muestra por pantalla el giro del avión en el eje z.

CU16 Movimiento eje x.

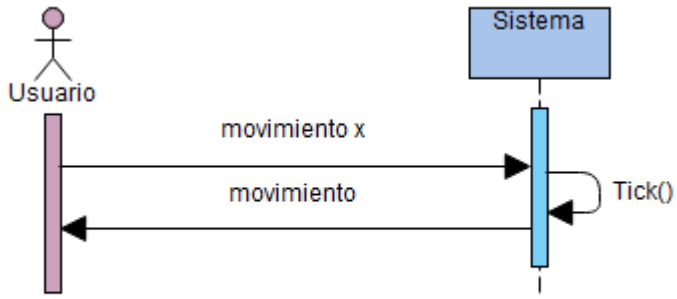

Figura 5.20: Diagrama de secuencia CU16.

- movimiento x
  - Actores : Usuario.
  - Responsabilidades : El usuario realiza un movimiento en el eje x desde los mandos del simulador.
  - Precondición : VisualPiperSeneca se está ejecutando y está en un vuelo.
  - Postcondición : VisualPiperSeneca recibe la orden de movimiento.
- Tick()
  - Actores : Sistema.

  - Responsabilidades : El sistema procesa los nuevos datos.
  - Precondición : VisualPiperSeneca ha recibido valores nuevos para el estado de la simulación.
  - Postcondición : El sistema procesa los nuevos datos sobre el estado de simulación.
- Rotación
  - Actores : Sistema.
  - Responsabilidades : El sistema procede a realizar un movimiento.
  - Precondición : VisualPiperSeneca ha procesado los datos sobre el movimiento del avión en la simulación.
  - Postcondición : El sistema muestra por pantalla el avance del avión en el eje x.

CU17 Movimiento eje y.

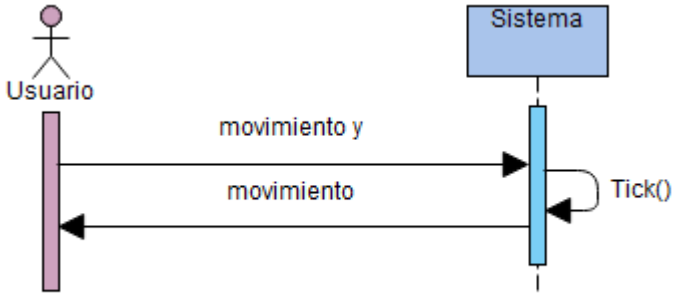

Figura 5.21: Diagrama de secuencia CU17.

- movimiento x
  - Actores : Usuario.
  - Responsabilidades : El usuario realiza un movimiento en el eje y desde los mandos del simulador.
  - Precondición : VisualPiperSeneca se está ejecutando y está en un vuelo.
  - Postcondición : VisualPiperSeneca recibe la orden de movimiento.
- Tick()
  - Actores : Sistema.
  - Responsabilidades : El sistema procesa los nuevos datos.
  - Precondición : VisualPiperSeneca ha recibido valores nuevos para el estado de la simulación.
  - Postcondición : El sistema procesa los nuevos datos sobre el estado de simulación.
- Rotación
  - Actores : Sistema.
  - Responsabilidades : El sistema procede a realizar un movimiento.
  - Precondición : VisualPiperSeneca ha procesado los datos sobre el movimiento del avión en la simulación.

- Postcondición : El sistema muestra por pantalla el avance del avión en el eje y.

CU18 Movimiento eje z.

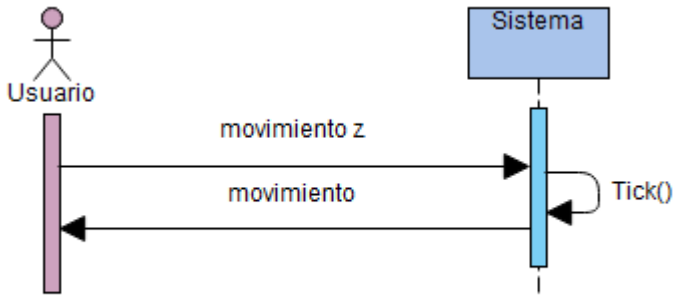

Figura 5.22: Diagrama de secuencia CU18.

- movimiento z
  - Actores : Usuario.
  - Responsabilidades : El usuario realiza un movimiento en el eje z desde los mandos del simulador.
  - Precondición : VisualPiperSeneca se está ejecutando y está en un vuelo.
  - Postcondición : VisualPiperSeneca recibe la orden de movimiento.
- Tick()
  - Actores : Sistema.
  - Responsabilidades : El sistema procesa los nuevos datos.
  - Precondición : VisualPiperSeneca ha recibido valores nuevos para el estado de la simulación.
  - Postcondición : El sistema procesa los nuevos datos sobre el estado de simulación.
- movimiento
  - Actores : Sistema.
  - Responsabilidades : El sistema procede a realizar un movimiento.
  - Precondición : VisualPiperSeneca ha procesado los datos sobre el movimiento del avión en la simulación.
  - Postcondición : El sistema muestra por pantalla el avance del avión en el eje z.

Capítulo 6

Diseño del sistema

Después de un análisis del sistema, explicaremos ahora los aspectos de diseño para el sistema.

6.1. Arquitectura del sistema

En este apartado se tratará de describir la arquitectura física y lógica del sistema, por tanto se explicará la estructura tanto de hardware y software, su funcionamiento y sus relaciones.

6.1.1. Arquitectura física

Este apartado hace referencia a los componentes necesarios para poder lanzar el sistema tanto hardware como software.

Respecto a los componentes hardware necesarios, al menos se debe disponer de un equipo capaz de ejecutar instrucciones x86, con una tarjeta gráfica medianamente potente.

Respecto a los componentes software se debe tener el sistema Windows instalado en el equipo.

6.1.2. Arquitectura lógica

Para definir la arquitectura lógica del sistema se ha usado un patrón de diseño, que según Pavón [9] es una solución a un problema en un contexto particular y permiten describir el esquema básico para estructurar sistemas y componentes.

Se ha seguido un patrón de diseño estructurado, según Peñalvo [10] este tipo de patrón describe la forma en que se componen los objetos, se cuidan de cómo las clases y los objetos se componen para formar estructuras mayores. Un patrón de diseño estructural, utiliza la herencia para componer interfaces o implementaciones.

Es por ello que se ha elegido este patrón, ya que a partir de clases bases debíamos generar componentes mayores, esto lo podremos ver con más detalle en el diagrama de implementación.

6.1.3. Diseño lógicos de datos

Debido a la naturaleza del sistema no ha sido necesario crear un diseño para los datos que el sistema procesa. La información que el sistema procesa, no es necesario almacenarla,

no es información sensible y toda la información que se genera, se usa en el tiempo de funcionamiento del sistema.

6.2. Modelo conceptual

Antes de explicar este apartado es necesario conocer algunos términos que son tratados:

La definición de clase según nos dice Moreno [11] es una abstracción de algún aspecto concreto y un conjunto de datos que comparten las mismas características.

El modelo conceptual según la definición de Robinson y Arbez [12] es una representación de la estructura y el comportamiento de un sistema utilizando un formato predefinido, Es decir es un tipo de diagrama que describe las relaciones entre las clases que conforman un sistema mostrando los componentes de las clases.

El modelo conceptual se ha creado utilizando el lenguaje UML que según Booch, Rumbaugh e Jacobson [13] se define como :'Un estándar diseñado para visualizar, especificar, construir y documentar software orientado a objetos'

Debido a la gran cantidad de atributos de las clases, no sea han añadido en el diagrama pero explicaremos algunos atributos, se adjunta a continuación el modelo conceptual de VisualPiperSeneca (véase figura 6.1).

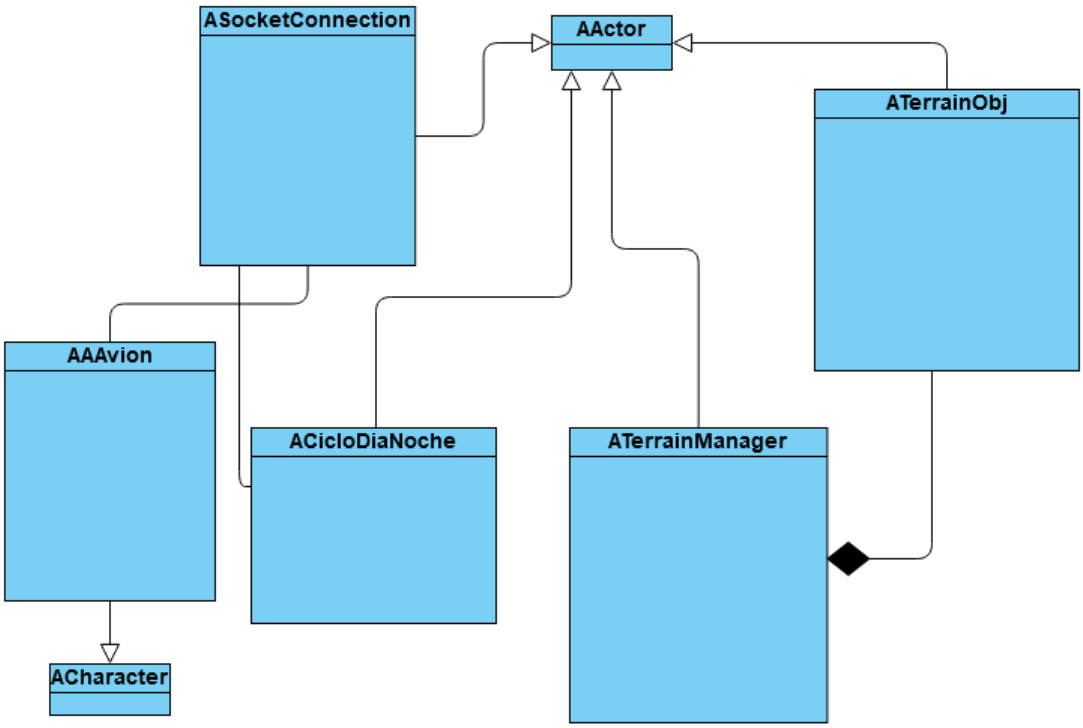

Figura 6.1: Diagrama conceptual.

6.2.1. Clase ACharacter

Esta clase representa un elemento manipulable en el sistema, es decir que se puede controlar, viene predefinido de la herramienta Unreal Engine 4 y facilita mucho la manipulación de objetos en el mundo.

6.2.2. Clase AAAvion

Clase que hereda de Acharacter, representa el avión en el sistema de visualización, hereda de Acharacter para poder ser poseído, es decir para poder moverlo en el mundo y ver el mundo desde su perspectiva, tiene diferentes métodos para aplicar movimiento en el mundo 3D, también tiene un detector de colisiones, muy útil para su uso en el simulador de vuelo, entre sus atributos destacamos un objeto de tipo cámara, que es la visión que tendremos en el mundo 3D

Tiene una relación de herencia con la clase ACharacter para facilitar su manipulación.

6.2.3. Clase AActor

Esta clase representa un objeto que puede recibir manipulaciones en el sistema, viene predefinido de la herramienta Unreal Engine 4 es muy útil a la hora de generar eventos que permitan la modificación de algún aspecto del mundo.

6.2.4. Clase ATerrainobj

Esta clase que hereda de AActor, representa un terreno en el mapa, es encargada de generar el objeto 3D en el sistema, entre sus atributos destacamos su ancho, largo y puntos de elevación, y los ficheros que generan el terreno.

6.2.5. Clase ATerrainManager

Clase que hereda de AActor, es la clase encargada de crear y destruir los terrenos en el sistema si se cumplen las condiciones necesarias, sus atributos más importante son un conjunto de datos que guarda referencias de los objetos ATerrainObj, y otro conjunto que almacena las posiciones de donde irán los terrenos para evitar recalcular en cada creación.

La relación que tiene con ATerrainObj, una composición, indica que ATerrainManager tiene entre sus atributos objetos de tipo ATerrainobj.

6.2.6. Clase ASocketConnection

Clase que hereda de AActor, esta clase representa la conexión con el simulador, es la clase encargada de crear una conexión y de conectarse con el simulador, recibir y enviar datos mediantes sockets. Esta clase se encuentra el protocolo de comunicaciones, mediante sockets de conexión, dos bus de datos y un conjunto de códigos se encarga de codificar y descodificar los datos para que el simulador pueda entenderlo y la comunicación sea correcta.

Entre sus atributos encontramos métodos para enviar y recibir datos, codificar y descodificar datos y muchas variables que luego envía a ACicloDiaNoche para llevar algunos cambios en el mundo.

6.2.7. Clase AcicloDiaNoche

Clase que hereda de AActor, aunque esta clase tenga ese nombre, que no nos engañe, no solo se encarga de hacer funcionar el ciclo día noche, también se encarga de cambiar en el mundo la hora y los aspectos climatológicos, entre sus atributos se encuentra un conjunto de variables para almacenar datos necesarios para poder realizar cambios de forma correcta y funciones que realizan los cambios en el mundo.

Tiene una relación con ASocketConnection debido a que recibe datos de esta clase y los almacenas en variables para poder llevar acabo los cambios, estas variables solo se actualizan dependiendo de los requerimientos recibidos de ASocketConnection y tras un cambio, expande hacia el mundo el cambio.

Capítulo 7

Implementación del sistema

En este capítulo explicaremos los aspectos de implementación del sistema, es decir, la estructura del código del sistema y los elementos usados para poder implementar el sistema.

7.1. Estructura del código

Tal y como se dijo anteriormente, el sistema se ha desarrollado usando Visual Studio, la aplicación ha generado una solución de forma automática, una solución realmente es un contenedor o carpeta que contienes los ficheros relacionados con el proyecto software junto a la información necesaria para poder compilar el proyecto. Un proyecto software es una colección o conjunto de ficheros fuente que al compilarlos genera un archivo ejecutable.

Respecto a la solución implementada, se llama VisualPiperSeneca (véase figura 7.1) y esta formada por tres carpetas con ficheros que explicaremos a continuación.

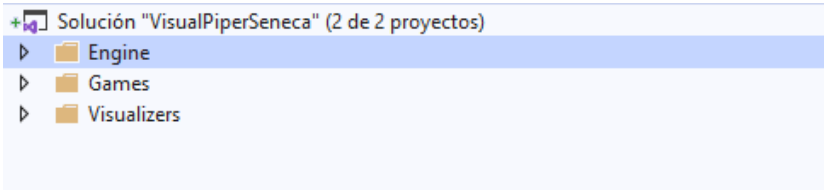

Figura 7.1: Estructura del código fuente del proyecto.

7.1.1. Carpeta Engine

Esta carpeta (véase figura 7.2) contiene datos que usará Unreal Engine 4 para poder compilar la solución de forma correcta, debido a que esta carpeta es realmente enorme y que viene por defecto de Unreal Engine 4 no tiene sentido explicarla, sin embargo es importante añadir que forma parte de la solución

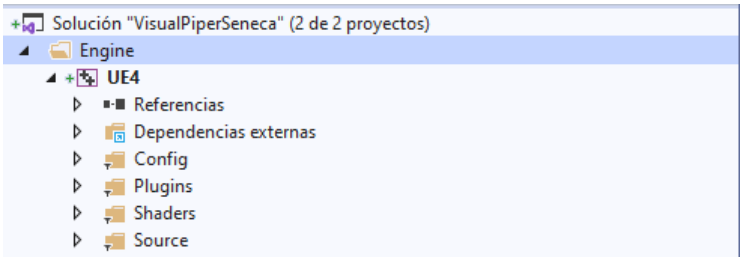

Figura 7.2: Contenido de la carpeta Engine.

7.1.2. Carpeta Games

Esta carpeta (véase figura 7.3) es la más importante puesto que contiene todo los códigos desarrollados para que la solución VisualPiperSeneca pueda funcionar de forma correcta.

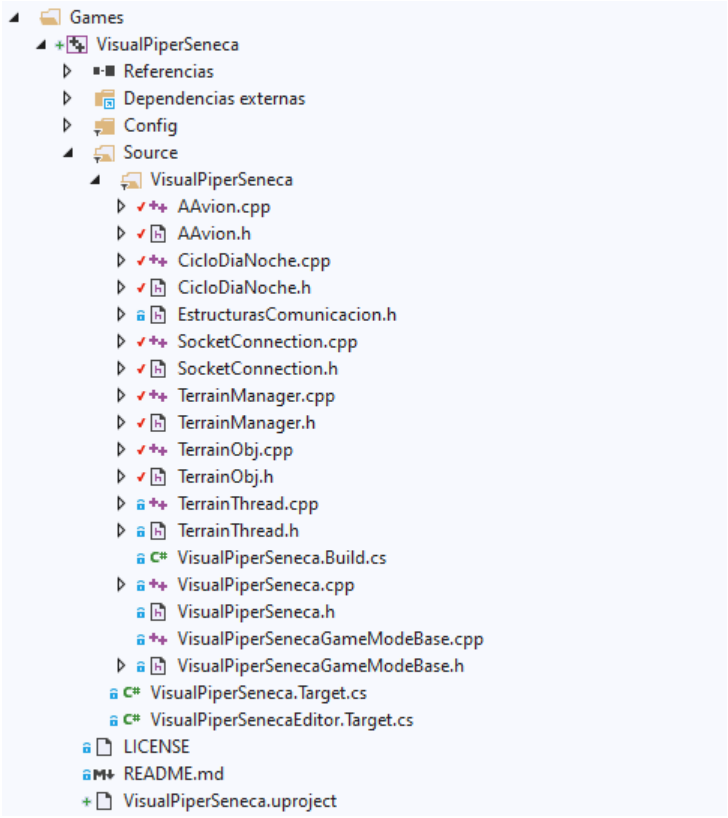

Figura 7.3: Contenido de la carpeta Games.

Carpeta Dependencia externas

Esta carpeta contiene enlaces a ficheros que podemos linkear en nuestros códigos desarrollados.

Carpeta Config

Carpeta que contiene archivos de tipo .ini y que tiene la configuración que queremos aplicar a Unreal Engine.

Carpeta Source/VisualPiperSeneca

Carpeta que contiene todos los ficheros fuentes y cabecera desarrollados en este proyecto.

Archivos AAvion: Estos ficheros contiene el código fuente desarrollado para la clase AAvion.

Archivos CicloDiaNoche: Estos ficheros contiene el código fuente desarrollado para la clase CicloDiaNoche, parte de sus funciones son también llamadas desde Blueprint de Unreal Engine.

Archivos SocketConnection: Estos ficheros contiene el código fuente desarrollado para la clase SocketConnection, representa la conexión y es el encargado de que el protocolo de comunicaciones funcione correctamente.

Archivos TerrainObj: Estos ficheros contiene el código fuente desarrollado para la clase TerrainObj, representan un terreno dentro del mundo.

Archivos TerrainManager: Estos ficheros contiene el código fuente desarrollado para la clase TerrainManager, es un objeto que se encarga de crear y destruir terrenos.

7.2. Elementos de Unreal Engine 4 usados

Juntos a los códigos anteriormente explicados y a los elementos que Unreal Engine nos aporta, se ha desarrollado un mundo que permite la generación de imágenes para el simulador. A continuación pasaremos a explicar los elementos que se han usado en la implementación del sistema desde Unreal Engine 4 (véase figura 7.4).

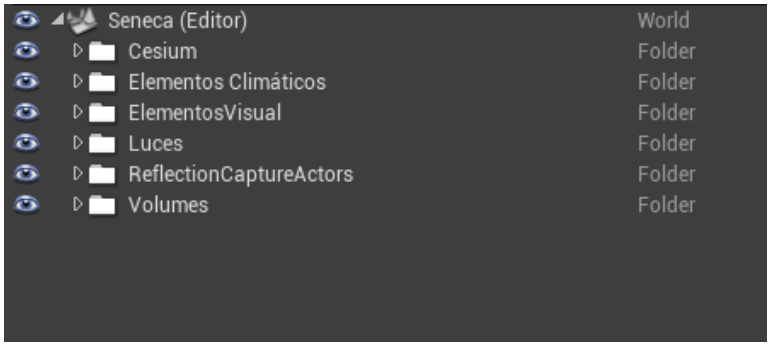

Figura 7.4: Carpeta de elementos usados en UE4.

Seneca es la representa el mundo creado. Es la unión de todos los componentes y objetos creados.

7.2.1. Carpeta Cesium

La carpeta Cesium (véase figura 7.5) contiene todos los elementos necesarios para poder utilizar el plugins Cesium.

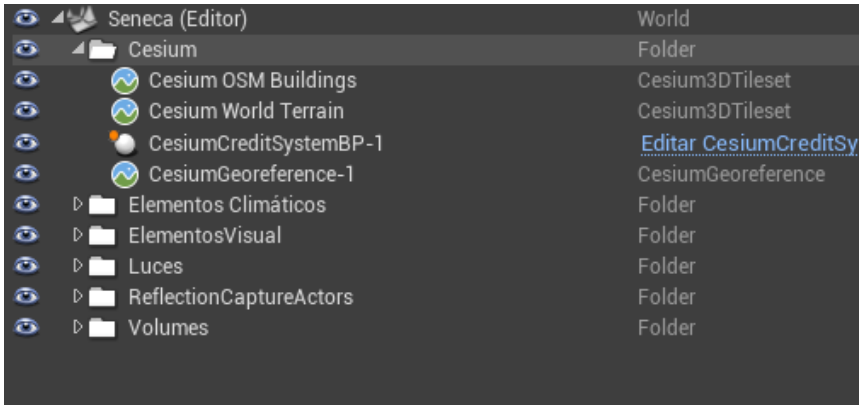

Figura 7.5: Carpeta Cesium.

Cesium es un potente plugin que fue añadido casi al final del desarrollo del sistema, permite cargar mapas reales de la tierra en el sistema, para añadirlo se tuvo que seguir las siguientes instrucciones de esta dirección `https://github.com/CesiumGS/cesium-unreal`.

Cesium OSM Building

Objeto que representa pequeños edificios solo es visible cuando Cesium esta activo.

Cesium World Terrain

Objeto que representa a la tierra, es decir todos los mapas de la tierra.

CesiumGeoreference-1

Objeto que permite movernos por el mundo de Cesium dando las coordenadas de latitud y longitud.

7.2.2. Carpeta Elementos Climáticos

La carpeta Elementos Climáticos (véase figura 7.6) contiene todos los objetos que interactúan con el mundo para generar los eventos climáticos y el ciclo día noche.

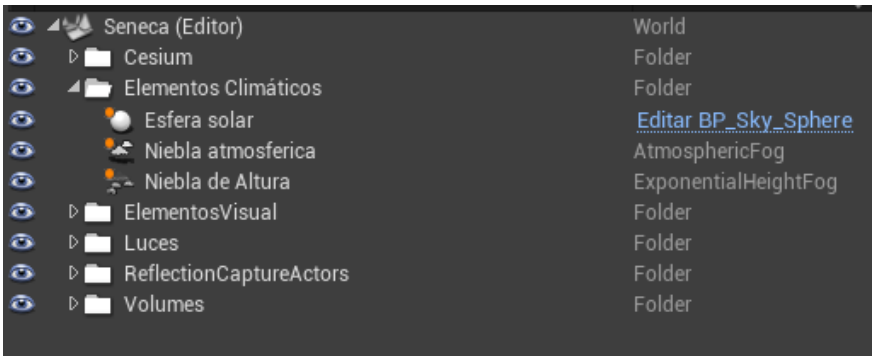

Figura 7.6: Carpeta Elementos Climáticos.

Esfera Solar

Objeto que representa el sol y el cielo, los movimientos de este objeto son los que permiten los ciclos dia y noche.

Niebla Atmosférica

Objeto que representa niebla poco intensa, según la intensidad que reciba de una función de ACicloDiaNoche, la niebla será más intensa o menos intensa.

Niebla de altura

Objeto que representa la niebla desde el cielo hasta el suelo, este objeto recibe datos de una función de ACicloDiaNoche que calcula según la altura del objeto AAvion la intensidad de la niebla.

7.2.3. Carpetas ElementosVisual

Carpeta que contienes los elementos desarrollados.(véase figura 7.7).

Figura 7.7: Carpeta ElementosVisual.

AAvion

Representación de un objeto de la clase AAAvion. Este objeto es la representación del avión en el mundo, es una clase desarrollada en C++ tiene unida unas luces de punto para que en la noche tengamos luz y no veamos la pantalla totalmente oscura.

CicloDiaNoche

Representación de un objeto de la clase ACicloDiaNoche. Este objeto permite la existencia de un ciclo dia y noche en el sistema. Desarrollada en C++.

SocketConnection

Representación de un objeto de la clase SocketConnection, este objeto es el encargado de crear la conexión mediante la IP de nuestra maquina, codificar los envíos y descodificar los datos recibidos, en este objeto se encuentra el protocolo de comunicaciones y fue desarrollado en C++.

TerrainManager

Representación de un objeto de la clase TerrainManager. Permite que cuando estamos usando la generación de terrenos en el mundo, se creen los terrenos necesarios y se eliminen cuando sea necesario, desarrollado en C++.

7.2.4. Carpetas Luces

Carpeta que contienes todo lo relacionado con las luces del mundo. (véase figura 7.8).

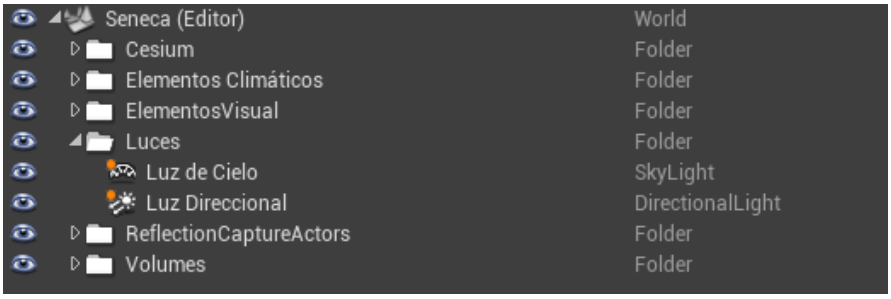

Figura 7.8: Carpeta Luces.

Luz de Cielo

Representación del sol en el mundo, es el objeto que se le aplican los cambios de posición y nos permite generar la transición entre dia y noche.

Luz Direccional

Representación de una pequeña luz en el mundo.

7.2.5. Carpetas Reflection y Volúmenes

Carpetas que contiene elementos para dotar al mundo de más realismo. (véase figura 7.9).

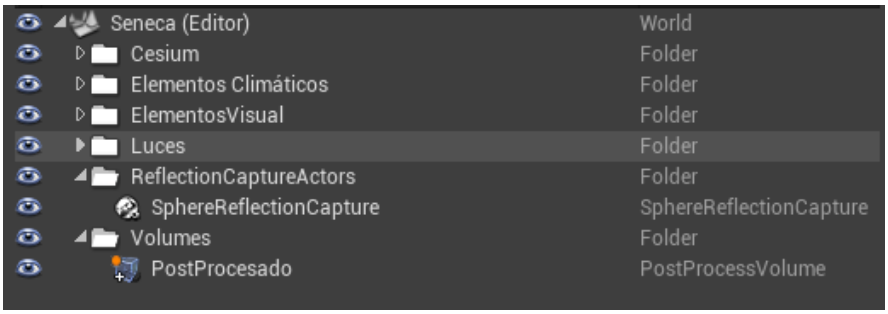

Figura 7.9: Carpetas Reflection y Volúmenes.

SphereReflectionCapture

Elemento que se encarga de hacer más realista la reflexión de las luces.

PostProcesado

Elemento que se encarga del postprocesado del mundo.

7.3. Protocolo de comunicaciones

El protocolo de comunicaciones es sin duda, la parte más compleja implementada, es por ello que merece ser explicada con más detalle.

Antes de comenzar a explicar es necesario entender que significa un protocolo, según la RAE (Real Academia Española) [14], la quinta definición a dia de la escritura de esta memoria dice que un protocolo es un conjunto de reglas que se establecen en el proceso de comunicación entre dos sistemas, por tanto nuestro protocolo de comunicación es un conjunto de reglas que se deben cumplir entre nuestro sistema y el simulador para funcionar correctamente.

Su funcionamiento es el siguiente:

El protocolo comienza a funcionar y lo primero que hace es obtener la IP de nuestra maquina y mediante el puerto 12000, abre una conexión mediante un socket que permite enviar y recibir datos.

Una vez abierta la conexión, para la comunicación con el simulador, se usa un conjunto de estructuras enviadas o recibidas mediante el socket anteriormente creado. El socket trabaja los datos mediante dos buses que deben tener de tamaño 1472 bytes ambos, Un bus se usara para el envío de datos y otro para la recepción, con esto evitamos perdidas de datos.

Las estructuras tienen un identificador único que permite saber que tipo de estructura llega y su tamaño, así con estos datos podremos ir enviando y recibiendo datos desde los buses de forma correcta sin pisar datos.

Durante un envió o recepción lo primero que debe estar en las primera parte del los buses de datos debe ser siempre la estructura cabecera(véase la figura 7.10).

```
typedef struct
{
    float                timestamp;
    unsigned short int length;
    unsigned char      status;

    char               chksum;
} RTX_header;
```

Figura 7.10: Estructura cabecera.

Las variables deben tener obligatoriamente los siguiente valores si queremos que el protocolo funcione correctamente:

lenght: Debe contener el número de bytes que se envía pero no debemos tener en cuenta los bytes de la cabecera.

status: Obligatorio debe contener el valor 21, ya que es un identificador de dirección de envió.

chksum: Aunque este valor llegue Dynamics ni lo usa, así que no tiene sentido su envió.

timestamp: Mismo caso que el anterior.

7.3.1. Envió de datos.

Debemos de crear una estructura cabecera y darle los datos explicados anteriormente, tras esto, debemos enviar al bus de datos de envió la estructura cabecera creada.

Tras el formateo y envió al bus de la cabecera, existe una estructura que se debe enviar al bus de forma obligatoria (véase figura 7.11) tras la cabecera para que el simulador pueda funcionar, en esta estructuras ira un contador interno que tomará valor en el primer envió de datos con el simulador y se lo tenemos que ir enviando en cada respuesta para comprobar la integridad.

Las estructuras obligatorias son las siguientes:

```
typedef struct
{
    unsigned char opcode;
    unsigned char length;
    unsigned char obj_id;
    unsigned char err_ref_id;


} RTX_rw_setup_res;
```

Figura 7.11: Estructura rw setup.

Las variables de la estructura rwSetup deben tener obligatoriamente los siguiente valores si queremos que el protocolo funcione correctamente:

opcode: Debe ser 0x26.

length: Tamaño de la estructura.

obj id: Debe tener el valor 1.

err ref id: Contador interno que tomará valor tras recibir el primer intercambio de datos.

Tras esto solo tendremos que ir enviado al bus de envió las demás estructuras(colisión, altura...etc) que aun no hemos enviado con sus respectivos valores para terminar el envió de datos.

7.3.2. Recepción de datos.

El socket se encargará de almacenar los datos recibidos para que no sean sobreescritos, después de almacenarlos. Estos datos se descodifican para que nuestro generador de imágenes pueda interpretar los valores de forma correcta.

La descodificación funciona leyendo los datos que se han almacenado del bus de datos, primero se lee los bytes correspondientes a la cabecera, que serán los primeros bytes que estarán almacenados, se leerá el tamaño de bytes que llega, y tras eso se avanza tantos bytes como el tamaño de la cabecera, esta posición contiene el tipo de estructura que sigue a la cabecera, se analiza esta estructura y posteriormente se avanza tantos bytes como el tamaño de la estructura, y así hasta terminar de leer todos los datos almacenados o se supere el número de bytes leídos según la cabecera.

7.4. Blueprint implementados

Los Blueprint permiten implementar funcionalidades sin programar, se han implementado varias funcionalidades usando Blueprint, se adjunta dos diagrama que muestran los

blueprint implementados (vease figuras 7.13 y 7.12)

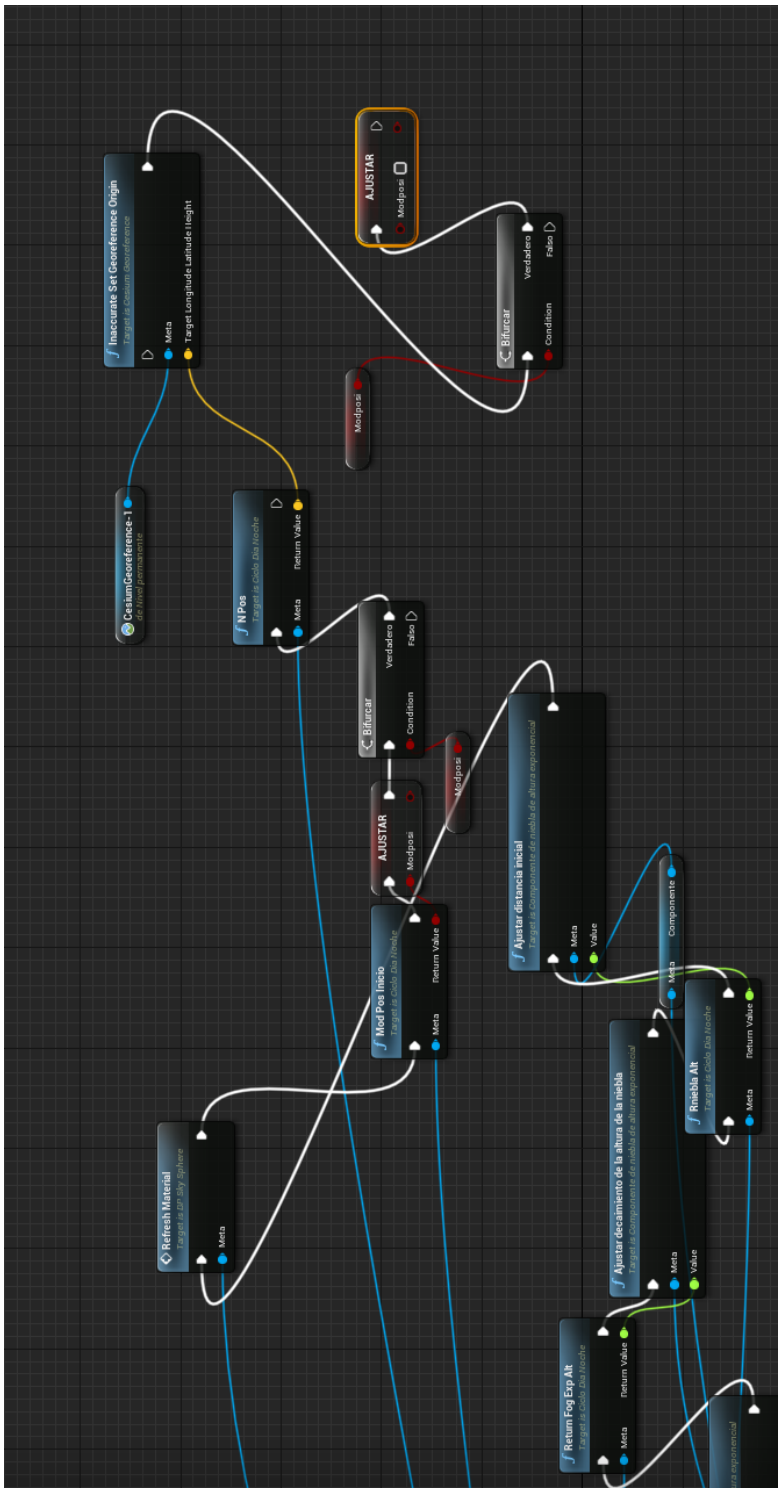

Figura 7.12: Blueprint Implementado 2º

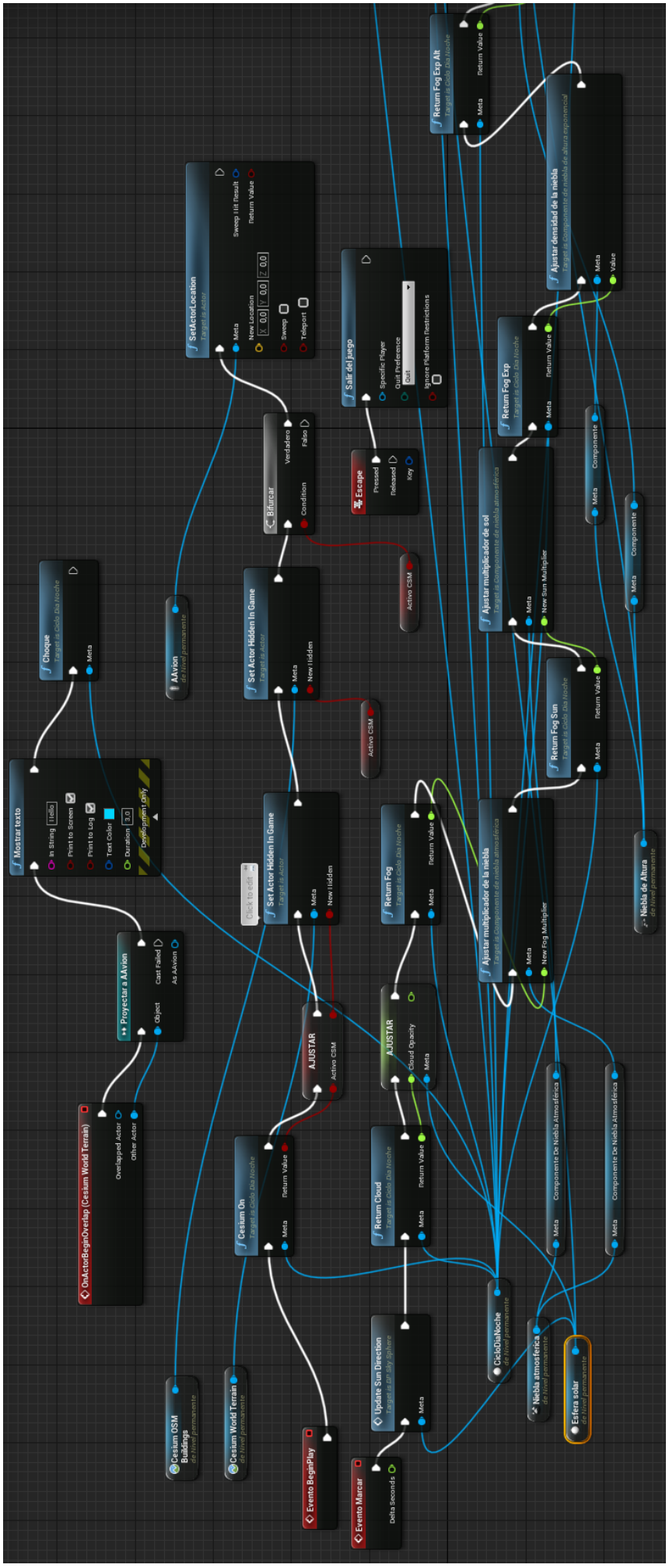

Figura 7.13: Blueprint Implementado 1º

7.5. Icono del sistema

Se ha diseño un icono gráfico sencillo para poder darle más identidad a el sistema (véase figura 7.14)

Figura 7.14: Icono del ejecutable del sistema.

Los iconos son muy importantes a la hora de crear e implementar un sistema, debido a que les dota de una identidad, lo que permite reconocer un sistema a partir del icono.

Capítulo 8

Pruebas de software.

En este capítulo se detallarán el conjunto de pruebas que se han realizado al software tras su desarrollo.

Según Glenford [15] las pruebas de software son el proceso de demostrar que no hay errores presentes.

Es por ello que era necesario las pruebas para comprobar el correcto funcionamiento. Todas las pruebas se realizaron tras terminar el desarrollo de VisualPiperSeneca ya que debido a su naturaleza no tenía sentido realizar pruebas en fases media.

8.1. Pruebas de caja negra

Este tipo de pruebas se enfocan en el comportamiento del programa, es decir las salidas que se producen según las diversas entradas posibles. Se ha dividido el funcionamiento del programa en varios bloques y a estos bloque se les ha sometido a un conjunto de pruebas, se adjunta un listado dividido en bloques con las pruebas realizada en el estados que se realizó y la salida que produce.

Bloque de lectura de configuración

Prueba 1 : No existe fichero de configuración.

Resultado 1 : El programa inicia con la configuración por defecto.

Prueba 2 : Existe fichero de configuración.

Resultado 2 : El programa inicia con los datos del fichero configuración.

Prueba 3 : Existe fichero de configuración pero con datos incoherentes.

Resultado 3 : El programa inicia con la configuración por defecto.

Bloque de conexiones

Prueba 1 : No estamos conectados a ninguna red.

Resultado 1 : El programa muestra un mensaje de error y se cierra.

Prueba 2 : Estamos conectados a una red distinta a la del simulador.

Resultado 2 : El programa inicia pero queda en espera infinita de recibir datos.

Prueba 3 : El programa inicia en la misma red que el simulador con la IP que le correspondiente.

Resultado 3 : El programa queda en espera de órdenes de IOS.

Bloque de envió de datos

Prueba 1 : Tras conectar con el simulador, realizamos un vuelo y enviamos datos sobre la simulación.

Resultado 1 : El programa recibe respuesta y muestra cambios en la pantalla.

Prueba 2 : Desconectamos durante un envió de datos.

Resultado 2 : El programa deja de recibir respuestas y queda en espera de volver a tener conexión.

Bloque de recepción de datos

Prueba 1 : Tras conectar con el simulador, realizamos un vuelo y esperemos a la recepción de datos sobre la simulación.

Resultado 1 : El programa recibe los datos, los procesa y muestras cambios por pantalla.

Prueba 2 : Desconectamos durante una recepción de datos.

Resultado 2 : El programa deja de recibir respuestas y queda en espera de volver a tener conexión.

Bloque de creación de ficheros de terrenos

Prueba 1 : No existe fichero RAW y si XML.

Resultado 1 : El programa no reconoce el terreno y no lo crea debido a a la falta del fichero.

Prueba 2 : No existe fichero XML y si fichero RAW.

Resultado 2 : El programa crea un terreno con los datos por defecto y las altura del fichero RAW.

Prueba 3 : Existen ambos ficheros y los tamaños no son excesivamente grandes 500*500.

Resultado 3 : Se crea el terreno con los tamaños indicados.

Prueba 4 : Existen ambos ficheros y el tamaño es desmesurado 50000*50000.

Resultado 4 : Error del programa debido a la falta de memoria.

Prueba 5 : Tras ejecutar el programa se llega a un punto donde un terreno debe ser creado según sus características.

Resultado 5 : Se crea el terreno de forma correcta.

Prueba 6 : Los ficheros no tienen la nomenclatura esperada.

Resultado 6 : El programa no crea ningún terreno.

Bloque de eliminación de terrenos

Prueba 1 : Se llega mediante un vuelo a un punto donde un terreno precargado debe ser borrado.

Resultado 1 : El programa elimina el terreno y envía a un fichero de logs con información sobre el terreno que fue y la posición que fue.

Bloque de Cesium

Prueba 1 : No estamos conectados a internet con Cesium inactivo en la configuración.

Resultado 1 : El programa no carga los datos de Cesium.

Prueba 2 : Estamos conectados a internet con Cesium activo en la configuración.

Resultado 2 : El programa inicia y carga los datos de terrenos de Cesium.

Prueba 3 : Estamos conectados a internet con Cesium inactivo en la configuración.

Resultado 3 : El programa inicia y carga los terrenos mediantes ficheros.

Bloque de Cambios en la simulación

Prueba 1 : Tras un inicio de vuelo se modifican las condiciones climáticas.

Resultado 1 : El programa recibe los datos y muestra por pantalla el cambio.

Prueba 2 : Durante el inicio de un vuelo se produce movimientos desde la cabina de pilotaje.

Resultado 2 : El programa muestra inmediatamente los cambio de posición.

Prueba 3 : Durante un vuelo, se pausa la simulación.

Resultado 3 : Tras unos segundos el programa se pausa y queda en espera.

Prueba 4 : Durante un vuelo, se ajusta la hora de la simulación.

Resultado 4 : El programa realiza el cambio de hora de vuelo y empieza un ciclo día noche desde esa hora a tiempo real.

Prueba 5 : Durante un vuelo, se decide desde el programa IOS cancelar el vuelo.

Resultado 5 : El programa cancela el vuelo y automáticamente queda en espera.

Prueba 6 : Tras montar todo el sistema, se inicia un vuelo desde IOS.

Resultado 6 : Tras unos segundos de recepción de datos, el programa comienza la simulación.

Prueba 7 : Se inicia un vuelo con una posición en concreto.

Resultado 7 : El programa inicia el vuelo en la posición indicada.

Bloque de interacción con el avión

Prueba 1 : Durante un vuelo procedemos a chocar con un terreno.

Resultado 1 : La simulación termina debido a el choque.

8.2. Ficheros logs y funciones debug

Existe en el código del programa VisualPiperSeneca, un conjunto de funciones que comprueba que ciertos valores son los esperados y si estos no lo fueran, envía los datos que presentan errores a un fichero de logs.

En este apartado no se explicarán estas funciones sino que se dará a conocer los ficheros de logs, ya que las salidas de dichas funciones van hacia estos ficheros que han sido de gran importancia a la hora de capturar y depurar errores. La localización de logs es la siguientes (vease figura 8.1).

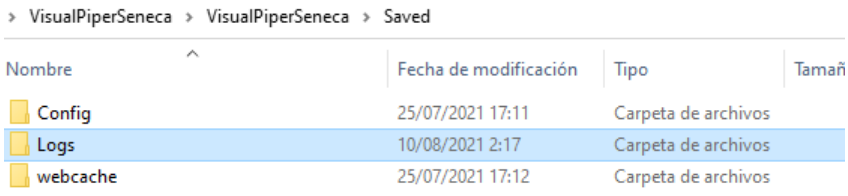

Figura 8.1: Localización de logs.

Un fichero logs es un fichero de texto que registra en el momento exacto eventos importantes que ocurrieron, es importante saber que se ha usado este tipos de ficheros, ya que actualmente con un vistazo pueden solventar muchos problemas, se muestra a continuación una imagen para que se pueda apreciar la estructura de estos ficheros (véase 8.2).

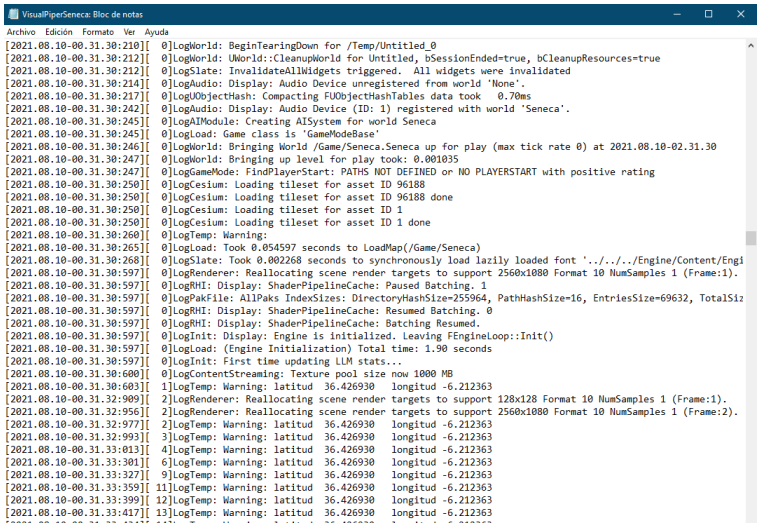

Figura 8.2: Fichero de logs.

8.3. Pruebas en vídeo

Debido a la naturaleza de VisualPiperSeneca, algunas pruebas fueron grabadas para demostrar que se comprobó su correcto funcionamiento, se adjunta una serie de enlaces con la descripción del vídeo. Los vídeos fueron subidos a la plataforma de vídeos YouTube.

- Prueba primera el avión se movía en base a los movimientos de la cabina y que el ciclo día noche funcionaba correctamente, enlace del vídeo `https://youtu.be/ehWgkoy2opQ`
- Prueba segunda el avión se movía en base a los movimientos de la cabina. Grabado desde la cabina, enlace del vídeo `https://youtu.be/joh_SmNjYtY`
- Prueba tercera en esta prueba se muestra el funcionamiento de los cambios en las condiciones climáticas. `https://youtu.be/Y8tKgOTJUA4`

Parte III

Epílogo

Capítulo 9

Conclusión

9.1. Experiencia

El desarrollo de este proyecto me ha proporcionado experiencia en varios sentidos.

En primer lugar, me ha permitido aprender y experimentar cómo organizar el desarrollo de un proyecto de una envergadura y complejidad a las que hasta ahora no estaba acostumbrado, al ocupar 10 meses de trabajo intensivo. Considero que esto es una valiosa experiencia que me servirá en el mundo laboral, donde hay muchos proyectos cuyo tamaño y complejidad requieren de varios meses de desarrollo, y por tanto tener experiencia previa en el desarrollo de proyectos de este tipo me permitirá abordar futuros proyectos de gran tamaño con mayor seguridad y confianza.

En segundo lugar, este trabajo me ha permitido adquirir experiencia con herramientas populares y demandadas en el mundo laboral, como Visual Studio y Unreal Engine. Haber aprendido a utilizar estas herramientas para crear un software tan complejo como el visualizador de un simulador de vuelo, creo que muestra una capacidad de aprendizaje y adaptación(que junto con el saber utilizar estas dos herramientas) que será muy valorada por futuros empleadores.

En tercer lugar, me ha proporcionado la experiencia de desarrollar una herramienta muy compleja y útil, con muchos componentes(cada uno con sus características propias) los cuales deben comunicarse de forma sincrónica y bidireccional de forma correcta para que todo funcione. Creo que desarrollar este visualizador de vuelo me ha proporcionado una experiencia de desarrollo de software muy parecida sino igual a la que habría obtenido trabajando en una empresa en el mundo real, donde se necesitan crear/actualizar herramientas complejas en sistemas que ya existen y están en funcionamiento, por lo que se deben crear o actualizar dichas herramientas siendo integradas dentro de un sistema de componentes complejos donde se debe tener en cuenta las características y funciones de cada componente individual para que todo el sistema funcione de forma correcta.

Por todo ello, creo que este TFG me ha proporcionado una experiencia muy valiosa y útil, a nivel personal y laboral, cuyo resultado a su vez me permitirá incluir este proyecto en mi CV y mis perfiles online, lo cual proporciona evidencia de que poseo capacidades que son muy útiles y demandadas en el mundo laboral.

9.2. Trabajo futuro

Aunque los objetivos propuestos se han cumplido y el desempeño y resultado final han sido satisfactorios, hay algunos aspectos que sería interesante incluir o mejorar:

- Hay algunos requerimientos del simulador, como el encendido de luces desde la cabina,

que no se han implementado, aunque estos requerimientos llegan como respuesta a VisualPiperSeneca, se podría añadir estudiando la posición de los botones de la cabina.

- Aumentar la eficiencia a la hora de generar terrenos mediante ficheros, debido a que esto toma demasiado tiempo, tanto que puede resultar molesto.
- Exportar a otras plataformas, de forma que no se necesite un equipo con Windows para poder ejecutar VisualPiperSeneca.
- Mejorar el texturizado de los terrenos, debido a que actualmente se necesitan materiales previamente creados en el editor de materiales de Unreal Engine 4.

Apéndice A

Manual de usuario

A.1. Introducción

La finalidad de este manual es, otorgar al lector el conocimiento suficiente para que pueda utilizar el módulo de visualización VisualPiperSeneca de forma correcta sin poseer conocimientos previos.

El manual puede ser dividido en 5 bloques:

- Requisitos.
- Realizar un vuelo.
  - Configuración de red.
  - Configuración del vuelo.
- Añadir terrenos nuevos a la visualización.
  - Fichero XML.
  - Fichero RAW.
  - Estructura del mapa formado por los terrenos.
  - Limitaciones.
- Uso de Cesium.
- Solución a errores comunes.

A.2. Requisitos

Para que el módulo se visualización VisualPiperSeneca pueda funcionar correctamente, es necesario:

- Procesador de cuatro núcleos AMD o Intel con al menos 2,5GHz de frecuencia
- Memoria RAM 4: GB.
- Tarjeta gráfica con al menos 2GB de VRAM.
- Debe ejecutarse bajo Windows.

A.3. Realizar un vuelo

Para la realización de un vuelo, antes debemos configurar correctamente las conexiones de red del equipo donde ejecutemos el software.

A.3.1. Configuración de red

Lo primero que debemos hacer es conectar el equipo al switch del simulador de vuelo PiperSeneca y configurar los ajustes de red del equipo, los siguientes pasos indican cómo configurar la red:

1. Abrir el panel de control del sistema operativo Windows.
2. Seleccionar la opción centro de redes y recursos compartidos.
3. En la zona izquierda de la ventana que se ha abierto, seleccionar Cambiar configuración del adaptador.
4. Seleccionar la red que está utilizando y hacer doble clic en ella.
5. Hacer clic en el botón propiedades, al hacer esto se nos abrirá una ventana con una lista, seleccionamos el apartado Protocolo de Internet versión 4(TCP/IP) y hacemos clic en propiedades.
6. En la ventana que se ha abierto seleccionamos introducir manualmente la dirección de IP, e introducimos la dirección IP que el programa Dynamics asigna para Visual, la mascara de subred se introducirá automáticamente tras introducir la IP y hacer clic en su campo correspondiente (IP por defecto para Visual 192.128.134.184), (en la figura A.1 podemos ver las ventanas de configuración de red).
7. Tras los pasos anteriores, hacemos clic en el botón aceptar y ya estará la configuración de red completada.

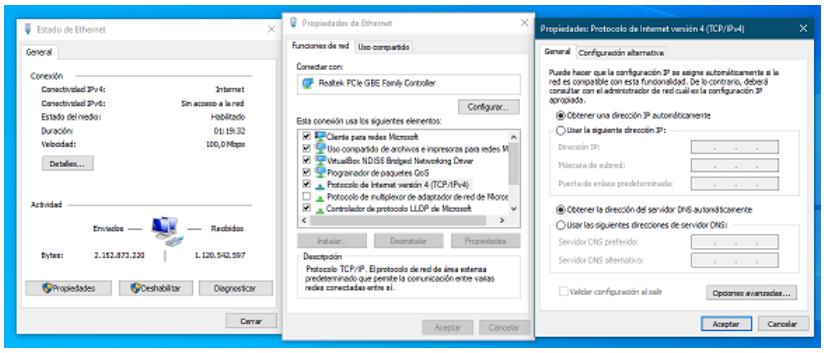

Figura A.1: Sucesión de ventanas de configuración de red.

Tras la configuración de red,debemos iniciar los demás equipos que conforman el simulador y ejecutar los programas correspondientes de cada equipo, tras comprobar que todo está conectado correctamente, ejecutaremos VisualPiperSeneca, tras su ejecución el programa quedará en espera de que la simulación se inicie mediante el programa IOS. Figura A.2

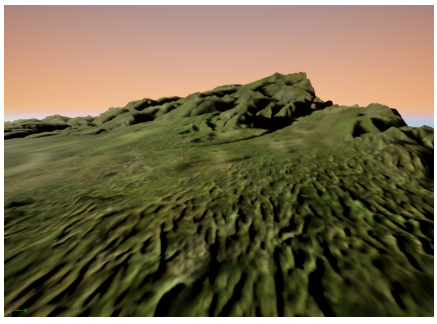

Figura A.2: VisualPiperSeneca en espera.

A.3.2. Configuración del vuelo

Iniciamos el programa IOS, podemos ver en la figura A.3 el aspecto de dicho programa, este programa nos permitirá iniciar vuelos así como controlar otros aspectos, tales como condiciones climatológicas y posición de inicio.

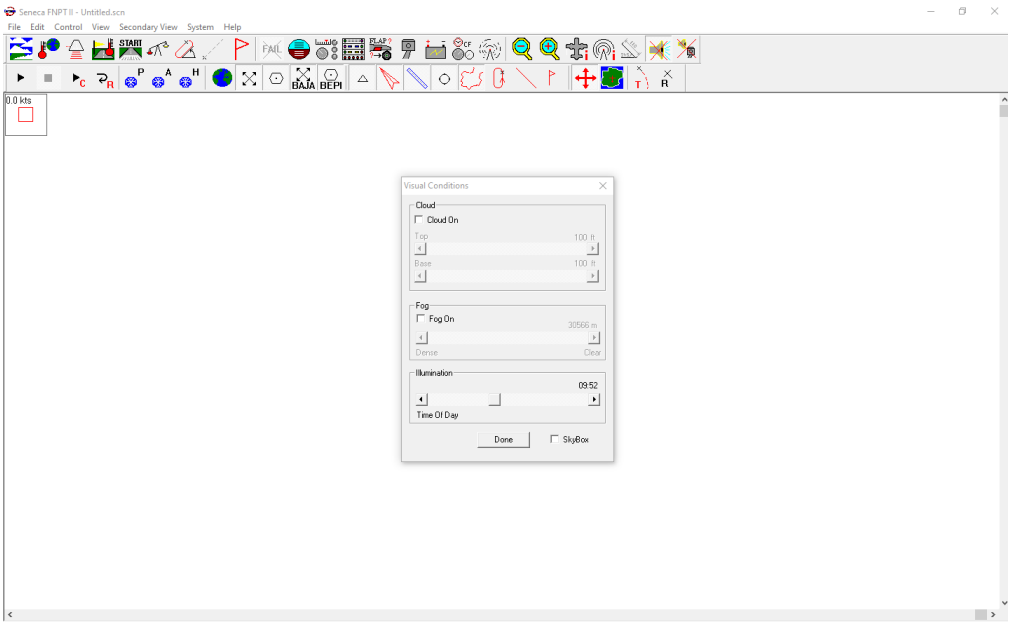

Figura A.3: Programa IOS.

Para iniciar un vuelo, primero debemos seleccionar la posición de inicio, para ellos debemos hacer clic en el botón START de IOS y se nos abrirá una ventana con información que podremos modificar para comenzar en distintos lugares o posiciones.

Después podremos comenzar el vuelo y modificar las condiciones climatológicas a nuestro gusto si fuera necesario, hacemos clic en el botón play, y comenzamos la simulación, esto hará que el simulador entre en pausa hasta pulsar un botón de la cabina de vuelo, antes de pulsar ese botón es cuando debemos cambiar las condiciones climatológicas desde IOS, para ello pulsamos el botón con la imagen de un paisaje nublado y se nos abrirá una ventana (véase figura A.4)

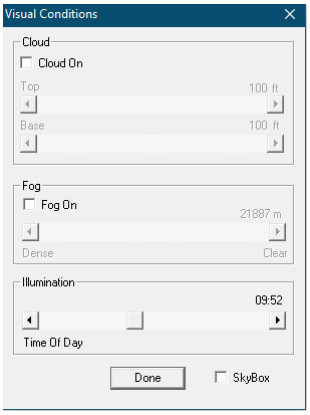

Figura A.4: Ventana condiciones climatológicas.

En esta ventana, podremos activar o desactivar las nubes y la niebla, así como ajustar

sus intensidades, también podremos elegir a la hora de comienzo del vuelo, la hora se verá posteriormente afectada debido a que se ha implementado un ciclo de día-noche.

Finalmente tras ajustar las condiciones climatológicas, entraremos en la cabina de vuelo y nos pondremos a los mando del simulador, pulsaremos el botón de pausa (véase figura A.5) para que el simulador salga del estado de pausa y comience el vuelo.

Para pausar un vuelo bastará con volver a pulsar el botón de pausa en la cabina de vuelo.

Figura A.5: Botón pausa.

Respecto a la finalización de un vuelo, este finalizará cuando hagamos clic en el botón stop desde IOS o cuando cometamos errores en el vuelo, tras finalizar un vuelo si queremos iniciar otro, solo tendremos que volver a ajustar la posición de inicio y hacer clic en el botón play.

A.4. Añadir terrenos nuevos a la visualización

Para añadir un nuevo terreno a la visualización, debemos generar dos ficheros, uno de extensión XML que almacenará información sobre las dimensiones del terreno y otro, de extensión RAW que almacenará datos de la elevaciones de los puntos del terreno, estos ficheros debemos añadirlos a una carpeta que debe tener el nombre de terrenos y además debe estar dentro de la carpeta de VisualPiperSeneca.

Es importante añadir que estos ficheros coincidan en sus tamaños, más adelante en este manual se explicará con más detalle en el apartado Limitaciones.

A.4.1. Fichero XML

Este tipo de fichero tendrá la estructura que se muestra en la figura A.6

```
<note>
        <ancho>2049</ancho>
        <alto>2049</alto>
        <lancho>10</lancho>
        <llargo>5</llargo>
        <distancia>1</distancia>
</note>
```

Figura A.6: Estructura fichero XML.

Donde cada parámetro indica las siguientes características del terreno:

- ancho : Número de puntos en el eje x
- largo : Número de puntos en el eje y
- lancho : Distancia entre los puntos del eje x
- llargo: Distancia entre los puntos del eje y
- distancia: Distancia que multiplica, la distancia en el eje x, y en el eje y, se recomienda dejar a 1.

Es importante darle valor a todos los parámetros, ya que puede dar lugar a incoherencias en el terreno.

A.4.2. Fichero RAW

Este tipo de fichero tendrá la estructura que se muestra en la figura A.7.

```
   y
X  6   22   2
   34  43  21
   87  54  56
```

Figura A.7: Fichero RAW de un terreno 3x3.

En este fichero almacenaremos los datos correspondientes a las alturas de cada punto del terreno, cada punto se almacenará en formato de entero corto sin signo(16 bits), generalmente los datos estarán estructurados en una matriz, donde las filas corresponden al eje x y las columnas al eje y, VisualPiperSeneca leerá los datos fila a fila.

A.4.3. Estructura del mapa formado por los terrenos

La idea de crear terrenos es, la de formar un mapa a partir de estos, para poder formar el mapa debemos indicarle a nuestros terrenos la posición en la que se situará en el mapa, para ello debemos indicar en el nombre de los ficheros XML Y RAW la posición en el mapa donde se situará el terreno.

La nomenclatura de nombre a seguir es la siguiente, X,Y.XML Y X,Y.RAW(véase figura A.8), donde X e Y son números pertenecientes a los números enteros e indicarán la posición en el mapa donde se situará el terreno.

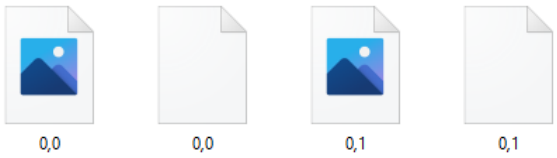

Figura A.8: Ficheros en la carpeta terrenos.

De esta forma, se irá creando un mapa en forma de cuadrícula, y dependiendo de los valores de X e Y iremos colocando terrenos(véase la figura A.9)

Y

X			
	Terreno -1.1	Terreno 0.1	Terreno 1.1
	Terreno -1.0	Terreno 0.0	Terreno 1.0
	Terreno -1.-1	Terreno 0.-1	Terreno 1.-1

Figura A.9: Estructura de mapa con 9 terrenos colocados de forma adyacente

A.4.4. Limitaciones

Es importante indicar las limitaciones a la hora de crear terrenos:

Los ficheros XML Y RAW deben coincidir en tamaños, que quiere decir esto, pues que si indicamos en nuestro fichero XML que tenemos un terreno de dimensión 100 x 100, debemos de tener tener un fichero RAW con 10000 enteros sin signos que indicarán la altura de cada punto.

También debemos evitar añadir terrenos muy grandes, es preferible crear dos terrenos más pequeños que un terreno muy grande, a parte de la cantidad de recursos que necesitaremos al crear un terreno de gran tamaño, sufriremos ralentizaciones e incluso puede que VisualPiperSeneca no se ejecute correctamente o se cierre de forma inesperada.

A.5. Uso de Cesium

Cesium es un complemento que se ha añadido a VisualPiperSeneca para generar mapas sin añadir terrenos.

Para usar Cesium debemos modificar el fichero Config.XML que se encuentra en directorio raíz de VisualPiperSeneca y ajusta la variable CESIUMOFF a 0, si queremos usar la generación de terrenos esta variable debemos ajustar el valor de esta variable a 1.

Es importante destacar que cuando usemos Cesium la generación de terrenos se desactivará de forma automática y que necesitaremos estar conectados a internet para que pueda funcionar de forma correcta.

A.6. Solución a errores comunes

Mi equipo no responde a las acciones de IOS

Pasaremos a verificar el estado de Dynamics. Si Dynamics nos muestra por pantalla, Visual coms Fails o Hardware coms Fails (véase figura A.10) nos indica que no se está comunicando con dichos programas, por tanto debemos revisar si el cable de red esta bien conectado y posteriormente comprobar que la IP de los equipos corresponde con las IP almacenadas en el archivo host de Windows en el equipo donde se ejecuta Dynamics, tras estas comprobaciones, reinicie los equipos y vuelva a intentarlo

Figura A.10: Mensaje de error de Dynamics.

En el caso de que Dynamics no muestre errores, reinicie los equipos y vuélvalo a intentar tras reiniciar.

En la simulación a veces la niebla se activa sola
Es normal, ya que según la altura a veces Dynamics envía un requerimiento de niebla que es de duración temporal.

VisualPiperSeneca se ha cerrado inmediatamente después de ejecutarlo
Esto puede ser debido a varias causas:

Una causa es que, en sus ficheros de terrenos, tiene un terreno de gran tamaño que hace que el equipo no pueda cargar en la memoria sus datos y el programa se cierre. Modifique o cree un terreno con tamaños más reducido o divida el terreno según sus preferencias.

Otra causa es que el equipo no esté conectado a ninguna red, VisualPiperSeneca intentará crear una conexión y al no disponer de red, este proceso fallará cerrando el programa de forma inesperada. Verifique que el cable de red está conectado y que dispone de conexión con el switch del simulador.

Apéndice B

Manual del programador

B.1. Introducción

La finalidad de este manual es, otorgar al lector el conocimiento necesario sobre los elementos de Unreal Engine 4, códigos fuentes escritos y Blueprint generados para ampliar la funcionalidad del módulo de visualización VisualPiperSeneca, se da por hecho que el usuario tiene un conocimiento medio sobre el lenguaje de programación C++ y del motor de diseño de videojuegos Unreal Engine 4.

El manual puede ser dividido en 8 bloques.

- Antes de comenzar
- Requisitos.
- Funcionamiento de VisualPiperSeneca.
- Tipos usados.
  - AAvion.
  - ATerrainManager.
  - ATerrainObj
  - ACicloDiaNoche.
  - ASocketConnection.
- Códigos Fuentes.
- Blueprint usados.
- Funciones debug.
- Aspectos finales.

B.2. Antes de comenzar

Aunque hay algunos aspectos que aunque se dan por sabidos, se hará una breve introducción.

Unreal Engine 4 es un motor de diseño de videojuegos que permite trabajar usando C++ y Blueprint, Epic Games [16] nos dice que Blueprint(Blueprint Visual Scripting) es 'Un sistema Scripting basado en el concepto de usar una interfaz basada en nodos para crear elementos desde Unreal Editor'.

Un Socket de conexión es según Quintana y Fecanin [17], 'Un Punto de Acceso al Servicio de Transporte y que las aplicaciones pueden usarlo para llevar a cabo comunicaciones a través de la red.'

B.2.1. Requisitos

Tendremos dos tipos de requisitos que cumplir a fecha de creación de este manual, para poder comenzar a desarrollar o ampliar VisualPiperSeneca, los requisitos de hardware, que son los características que deben cumplir el hardware de nuestro equipo y los requisitos de software, que son el software necesario para poder trabajar.

Los requisitos de hardware son los requisitos para que el editor de Unreal Engine 4 pueda funcionar correctamente. Según Epic Games [16] son los siguientes:

- Procesador de cuatro núcleos AMD o Intel con al menos 2,5GHz de frecuencia
- Memoria RAM : 8GB.
- Tarjeta gráfica con al menos 2GB de VRAM.

A continuación especificaremos los requisitos de software, que son los siguientes:

- Microsoft Visual Studio instalado en el equipo
- Unreal Engine instalado en el equipo.
- Sistema Operativo : Windows 7 en adelante

Si no cumplimos los requisitos de software, debemos instalar en nuestro equipo Unreal Engine 4 que ya viene con Microsoft Visual Studio integrado, lo podremos encontrar en la página de Epic a fecha de la creación de este manual, lo encontraremos en el siguiente enlace `https://www.unrealengine.com/en-US/download`

Tras la instalación procederemos a copiar la carpeta del proyecto a la carpeta de proyectos de Unreal Engine 4, usualmente esta carpeta se encuentra en la carpeta 'Documentos' del sistema operativo a no ser que se haya indicado otra carpeta en el proceso de instalación de Unreal Engine 4.

Tras la copia, procederemos a ejecutar Unreal Engine 4, para comprobar que todo fue correcto y que el motor detecta nuestro proyecto. En la figura B.1 podemos ver la apariencia general del Unreal Engine 4.

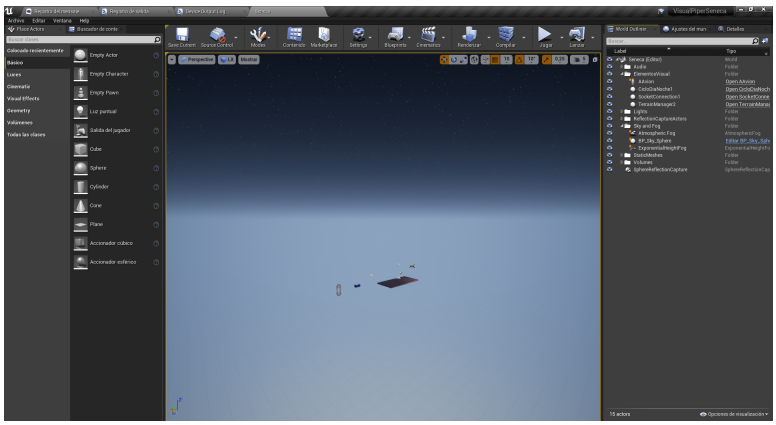

Figura B.1: Editor de Unreal Engine 4 en ejecución.

B.3. Tipos desarrollados

Antes de comenzar a explicar los tipos desarrollados, es importante que conozcamos dos tipos que incorpora Unreal Engine 4 debido a que estos tipos se usan de clase base para los demás:

El tipo Character, Epic Games [16] nos dice que abarca los elementos que en algún momento precisen movimiento.

El tipo AActor, Epic Games [16] nos dice que abarca los elementos que pueden aparecer en el mundo y realizar diversas acciones.

Tras conocer las clases bases, se explicará los tipos desarrollados que se han usado en el proyecto para dar una visión de su comportamiento, más adelante se profundizará en el código fuente pero antes es importante conocer ciertos aspectos. Para ver los tipos y elementos usados (véase la figura B.2).

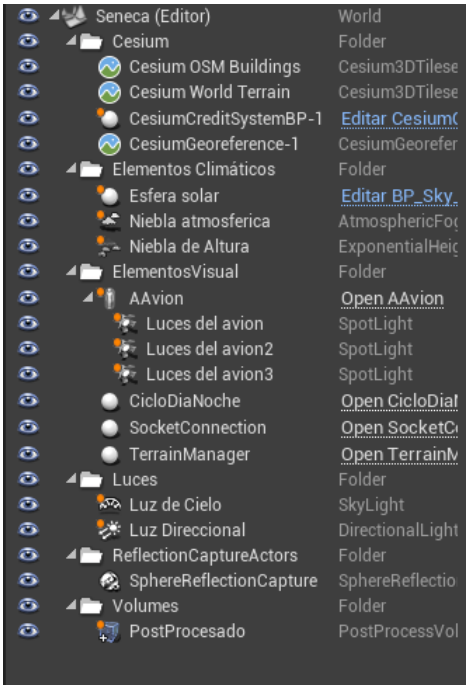

Figura B.2: Elementos de UE usados en el proyecto.

B.3.1. Clase AAAvion

La clase AAAvion hereda de la clase Character, esta clase representa el avión, la vista que tendremos de VisualPiperSeneca será desde la perspectiva de una instancia de esta clase, también es la encargada de calcular la distancia entre su posición y el punto del terreno que estemos atravesando.

Tiene luces asociadas pero estas por decisión de diseño no tienen la intensidad suficiente, solo sería necesario ajustarlo para tener la intensidad deseada.

B.3.2. Clase ATerrainManager

La clase ATerrainManager hereda de la clase AActor, esta clase es la encargada de gestionar la creación y eliminación dinámica de terrenos cuando se cumplen las condiciones de

distancia entre avión y terrenos, creará elementos de tipo ATerrainObj que son la representación de un terreno.

B.3.3. Clase ATerrainObj

La clase ATerrainObj hereda de AActor, esta clase es la encargada de leer los ficheros con los datos de los terrenos.

B.3.4. Clase ACicloDiaNoche

La clase ACicloDiaNoche hereda de la clase AActor, esta clase recibe los datos las condiciones climatológicas y junto con el Blueprint del nivel recrea las condiciones climatológicas.

B.3.5. Clase ASocketConnection.

La clase ASocketConnection. hereda de la clase AActor, esta clase es la encargada de conectarse con Dynamics, recibir y enviar datos mediante Socket UDP, para ello coge la dirección IP del equipo en el que se este ejecutando VisualPiperSeneca.

B.4. Explicación de funciones más relevantes

En esta sección se incluirán algunas definiciones de funciones presentes en los ficheros de cabecera de cada clase y se explicará las funciones más relevantes de los tipos desarrollados pero antes de incluir los ficheros explicaremos dos funciones comunes en todos los ficheros.

La funciones BeginPlay() y Tick() están presente en todos los tipos generados. BeginPlay() se ejecuta al crear una instancia del tipo y se ejecuta una sola vez. Tick() se ejecuta cuando el tipo ha terminado de crearse y se ejecuta en cada fotograma.

A continuación, se adjunta los ficheros.

B.4.1. Fichero AAvion

```
1
2  UCLASS()
3  class VISUALPIPERSENECA_API AAAvion : public ACharacter
4  {
5    GENERATED_BODY()
6
7  public:
8    AAAvion();
9    virtual void Tick(float DeltaTime) override;
10   virtual void SetupPlayerInputComponent(class UInputComponent*
       PlayerInputComponent) override;
11
12 protected:
13   virtual void BeginPlay() override;
14
15 private:
16   UCameraComponent* Cam;
17   USpringArmComponent* brazo;
18   APlayerController* Camara;
19   void actualizarEstado();
20   float DistanciaZ(FVector actual);
21 };
```

void actualizarEstado()

La función actualizarEstado(), actualiza la posición y estado de giro del objeto AAAvion mediante dos objeto de tipo FVector llamados pos y rotar, estos vectores son actualizados en cada fotograma y se llama a las funciones setActorLocation y SetActorRelativeRotation

para darle valores al objeto AAAvion. las funciones anteriormente nombras estas funciones son heredadas de Character.

float DistanciaZ(FVector)

Esta función se usa para calcular la altura del objeto AAAvion sobre el terreno que esta sobrepasando, luego esta distancia será usada para calcular choque con el terreno
.

B.4.2. Fichero SocketConnection

```
1  void recibirDatosSocket();
2  void enviarDatosSocket();
3  void Descodificar(int bytesleidos);
4  void Codificar();
5  int setRway(RTX_rw_setup_req* rw);
6  int setObjectSph(RTX_object_sph_req* sobj);
7  int setSwitch(RTX_switch_req* swiobj);
8  int setFog(RTX_fog_req* foj);
9  int setDayH(RTX_time_of_day_req* dia);
10 int setCloud(RTX_cloud_layer_req* nube);
11 int setEye(RTX_eye_offset_req* ojo);
12 int setRot(RTX_loc_rot_req* rot);
13 int setGs(RTX_gswitch_req* gs);
14 int setSky(RTX_sky_req* sky);
15 int setTfb(RTX_tfb_vtx_req* tfb);
16 int objaux(RTX_object_res* obj);
17 int objRes(RTX_coll_res* res);
18 int setTerrainFed(int* length,uint8* punteroBufferCod);
19 int setColision(int* length, uint8* punteroBufferCod);
20 int setSearchRes(int* length, uint8* punteroBufferCod);
21 int setupRW(int* length, uint8* punteroBufferCod);
22 int setObjRes(int* length, uint8* punteroBufferCod);
23 int AcknowledgeRes(int* length, uint8* punteroBufferCod);
```

void Descodificar(int bytesleidos)

Una de las funciones más importantes ya que se encarga de leer el buffer de datos recibidos, según los datos recibidos y la cabecera que se recibe se encarga de llamar a las funciones de tipo set para que todos los cambios sean reflejados.

void Codificar()

Otra función de gran importancia, ya que se encarga de recopilar los datos del objeto AAvion, de los terrenos, posiciones y los codifica en el buffer de envió para que Dynamics pueda entenderlo.

Funciones cuyo nombre tiene el prefijo set

Estas funciones son las encargadas tras descodificar, de almacenar en estructuras los cambios necesarios y trasmitirlos para poder realizarlos.

B.4.3. Fichero TerrainManager

```
1  FString DevuelveTerrenoActualString();
2  void DevuelveTerrenoActualInt(int32*,int32*);
3  double DistanciaTerrenosActivos(FString posicion);
4  void EliminaTerrenosActivos(ATerrainObj*);
5  void EliminadorTerrenosTick();
```

```
6   void CreadorTerrenosTick();
7   double DistanciaPuntoMedio(ATerrainObj*);
8   double DistanciaTerrenosNoActivo(int, int);
9   bool EsLimite(int,int);
10  void CalcularPos(FString fichero, int x, int y);
11  int ajustarPosicionMapaYNegativa(int x, int y);
12  int ajustarPosicionMapa(int x, int y);
13  int ajustarPosicionMapaNegativa(int x, int y);
14  int ajustarPosicionMapaY(int x, int y);
15  int lecturaXmlLargo(const FXmlNode* ficher);
16  int lecturaXmlAncho(const FXmlNode* ficher);
17  void InsertarDatosTer(FString fichero, int x, int y);
```

CreadorTerrenosTick()

Esta función es la encargada de generar los terrenos que el usuario tenga en la carpeta terrenos según la posición del avión. Recorre los ficheros de terrenos y en base a donde este el objeto AAvion genera automáticamente los terrenos.

EliminadorTerrenosTick()

Función que elimina los terrenos por lo que hallamos pasado cuando estemos a una distancia considerable de ellos.

B.4.4. Fichero CicloDiaNoche

```
1   UFUNCTION(BlueprintCallable)
2     float returnCloud();
3   UFUNCTION(BlueprintCallable)
4     float returnFog();
5   UFUNCTION(BlueprintCallable)
6     float returnFogSun();
7   UFUNCTION(BlueprintCallable)
8     float returnFogExp();
9   UFUNCTION(BlueprintCallable)
10    float returnFogExpAlt();
11  UFUNCTION(BlueprintCallable)
12    float RnieblaAlt();
13  UFUNCTION(BlueprintCallable)
14  bool CesiumOn();
15  void anularFog();
16  void CalcularFog();
17  void CalcularAltFog();
18  bool LecturaXml(FXmlNode* node);
```

Funciones cuyo nombre tiene el prefijo return

Todas estas funciones son las encargadas de llevar todos los cambios que tengan que ver con las condiciones climáticas y estado del día, si nos fijamos, todas estas funciones tienen encima de su declaración UFUNTION(BlueprintCallable) este identificador es muy importante, ya que permite llamar a dicha función en los Blueprint.

CesiumOn()

Función que determina, tras leer el fichero de configuración que tipo de terrenos se usarán, si los generados por el usuario o cargando los datos de terrenos de Cesium.

B.5. Blueprint usado

Se ha usado un Blueprint de nivel, su función es gestionar la activación o no de Cesium según el fichero de configuración y también es él encargado de la gestión de las condiciones climáticas y hora del día durante la simulación según los datos que se reciban mediante la clase ASocketConnection, se adjunta la estructura de dicho Blueprint (véase las figuras B.4 y B.3).

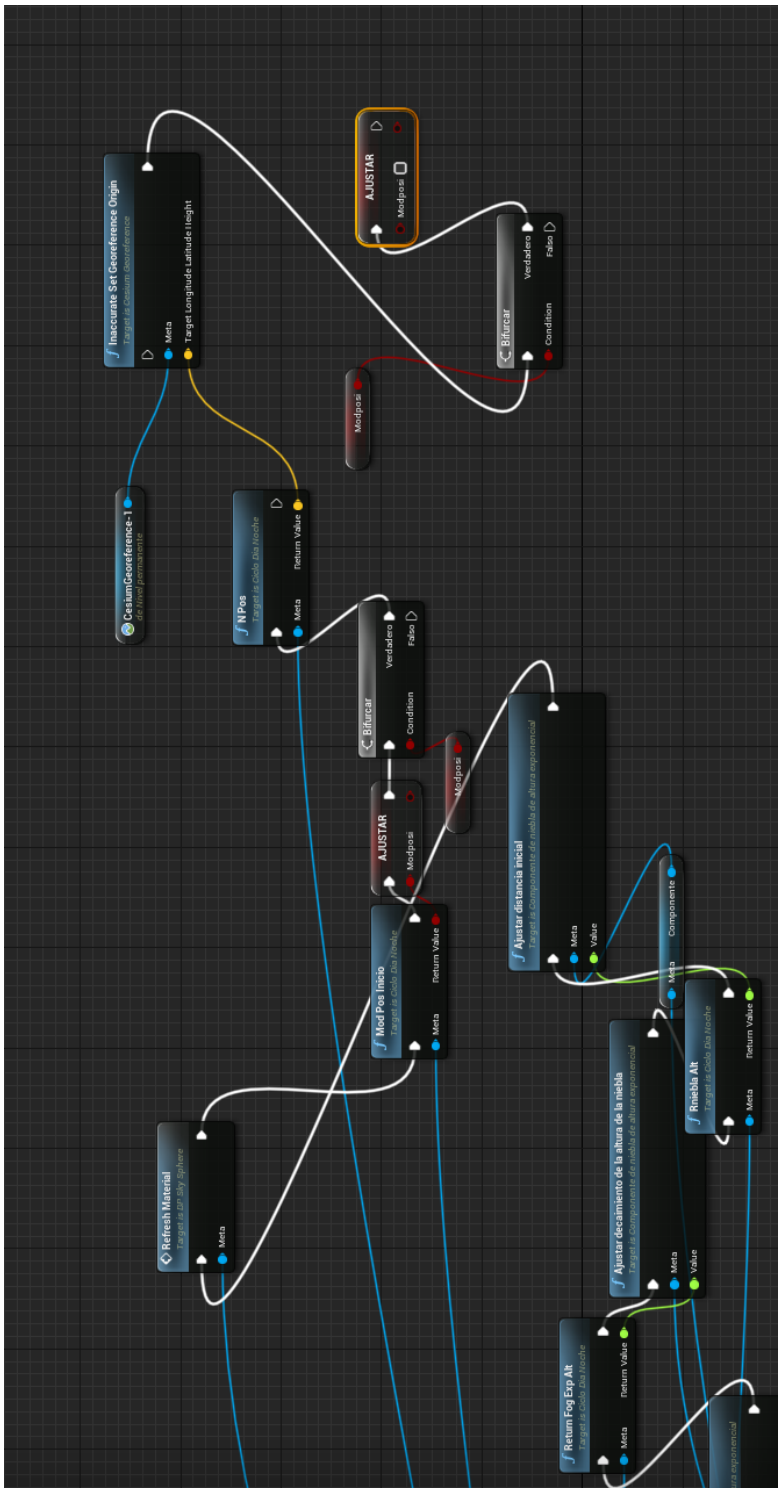

Figura B.3: Blueprint Implementado 2º

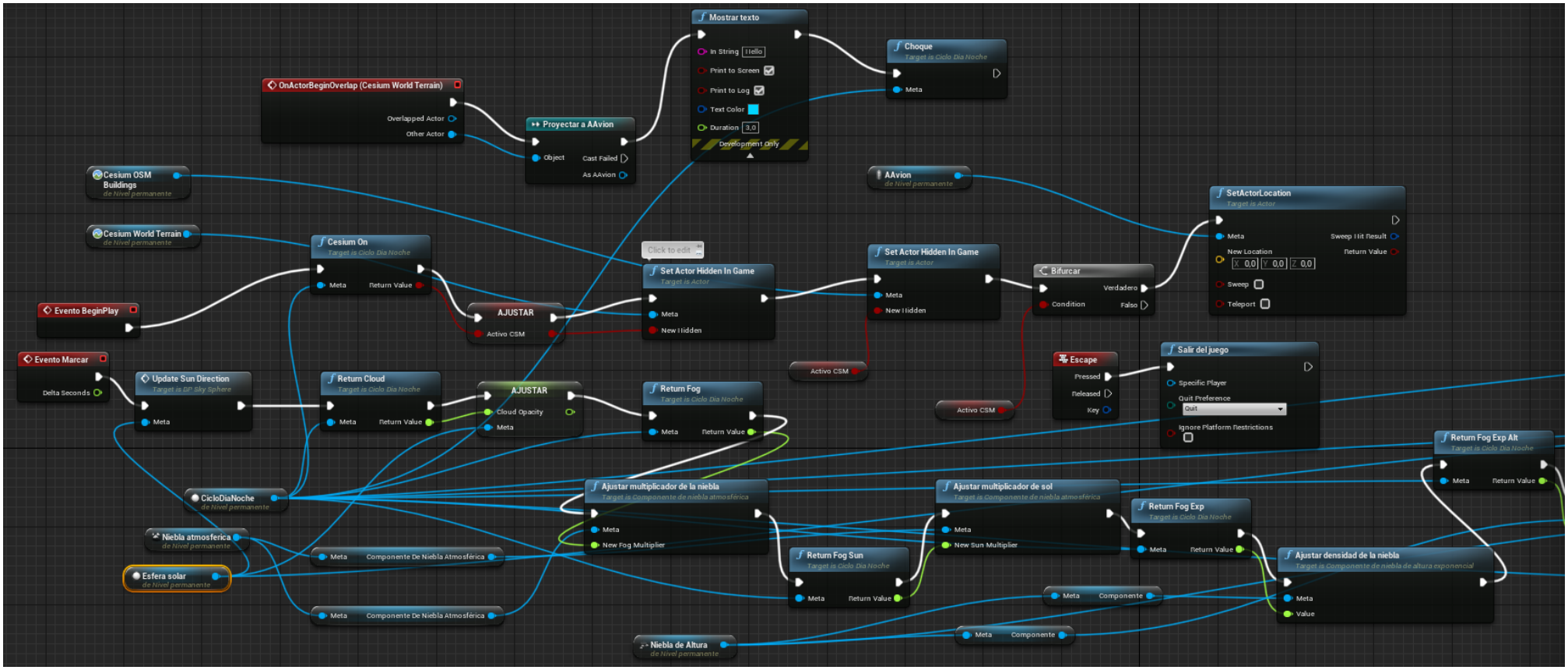

Figura B.4: Blueprint Implementado 1º

B.6. Funciones debug

Estas funciones, son funciones encargadas de imprimir en pantalla resultados, comprobar estados e imprimir eventos en los ficheros de logs para comprobar que todo funciona de forma correcta. Es importa añadir también que todas las funciones que recibe datos de Dynamics tienen comentado una linea de código que envía el dato recibido a un fichero de logs, dichas lineas están comentadas y marcadas como debug debido a que de no estarlo, se crearía un fichero de logs de gran tamaño.

Para poder activarlas las funciones debug tan solo es necesario llamar a estar funciones donde el programador considere que es necesario, por ejemplo tras una eliminación de terreno, o quitar el comentario de la función que recibe datos oportuna que se quiera comprobar.

Se adjunta un fragmento de una de las funciones debug que permite imprimir los contenedores de datos junto a su comentario que indica que es debug para poder diferenciarla de las demás.

```
1  //Debug Fun
2  void ATerrainManager::PrintMAP(){
3    for (const TPair<int, int>& pair : LimitesTerrenosSup)
4    {
5      UE_LOG(LogTemp, Warning, TEXT("valor MAP terreno superior key %i valor %i"
       ), pair.Key, pair.Value);
6
7    }
8                            .
9                            .
10
11   }}
```

B.7. Aspectos finales

Es importante añadir que el código esta debidamente comentado y que muchas funciones tienen en un comentario su funcionalidad, no se ha añadido más datos sobre funciones debido a que no es el objetivo que busca este manual.

MUY IMPORTANTE

Se recomienda encarecidamente realizar una copia de seguridad antes de tocar cualquier parte del código, en especial los ficheros SocketConnection. Todas las funciones de estos archivos tienen lineas necesarias para que pueda iniciar y funcionar la simulación, si ciertas lineas de código se modifican en algún aspecto puede dar lugar a que no se conecte el simulador, no inicie la simulación o Dynamics este enviando errores sin parar (véase figura B.4). Estas lineas tienen un comentario avisando de que no se deberían tocar. Es más, no se deberían tocar debido a que el protocolo de comunicación con el simulador exige que sea de la forma en la que ya está realizado, no me hago responsable de una mala modificación.

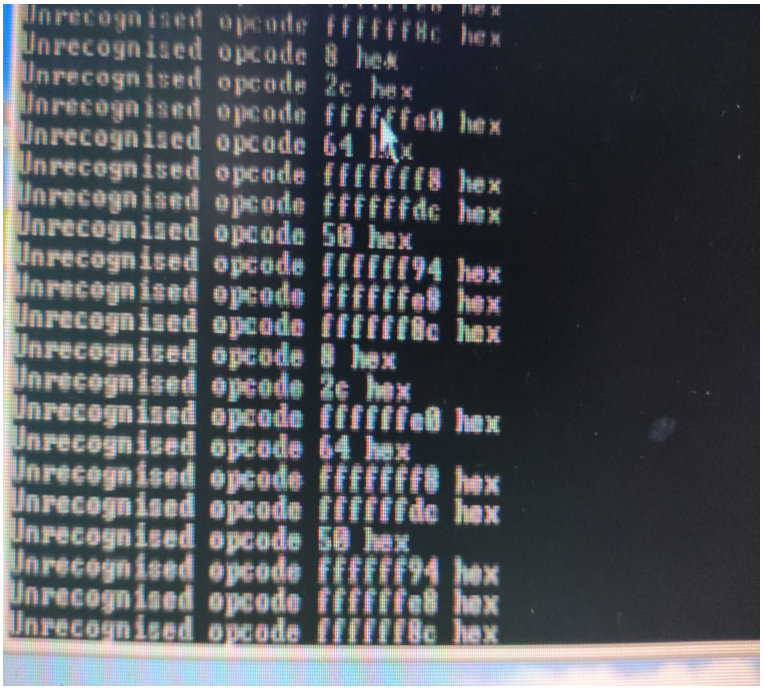

Figura B.5: Errores debido a una mala manipulación de código de comunicación.

Apéndice C

Manual de instalación

C.1. Introducción

La finalidad de este manual es otorgar al lector el conocimiento necesario para que VisualPiperSeneca sea instalado en el equipo y funcione correctamente.

C.2. Requisitos

Para poder instalar VisualPiperSeneca debe tener en su equipo al menos 2GB libres.

C.3. Instalación

Lo primero que debemos hacer es posicionar la carpeta de VisualPiperSeneca en un directorio del cual el usuario tengas permisos de lectura, debido a que VisualPiperSeneca realiza lectura de diferentes tipos de ficheros.

Posteriormente debemos comprobar que en el directorio de VisualPiperSeneca existe la carpeta terrenos y un fichero llamado config.xml(véase figura C.1)

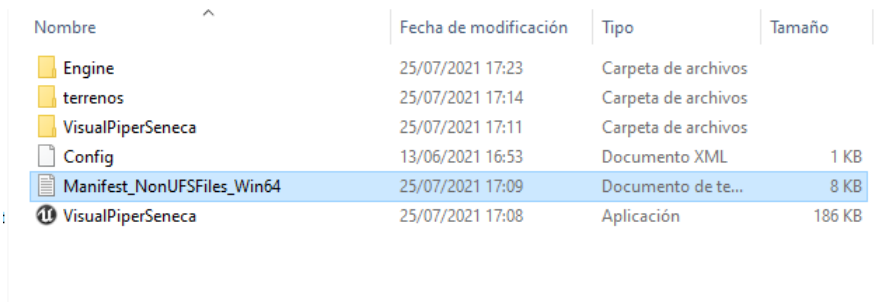

Figura C.1: Contenido de la carpeta VisualPiperSeneca.

Si no existen debemos crearlos. Para crear una carpeta pulsamos click derecho en la carpeta raíz de VisualPiperSeneca sin seleccionar ningún elemento y seleccionamos la opción crear carpeta nueva, la nombraremos 'terrenos'. Para crear el fichero config.xml, abrimos un procesador de texto, por ejemplo el bloc de notas y creamos un fichero nuevo e introducimos lo siguiente: (véase figura C.2)

```
<CONFIG>
        <CESIUMOFF>0</CESIUMOFF>
</CONFIG>
```

Figura C.2: Contenido del fichero config.xml

Al guardar lo guardamos con el nombre de config.xml y lo almacenamos en la raíz de VisualPiperSeneca.

Lo siguiente que debemos hacer es conectar el equipo al switch del simulador de vuelo PiperSeneca y configurar los ajustes de red del equipo, los siguientes pasos indican como configurar la red.

1. Abrir el panel de control del sistema operativo Windows.
2. Seleccionar la opción centro de redes y recursos compartidos.
3. En la zona izquierda de la ventana que se ha abierto, seleccionar Cambiar configuración del adaptador.
4. Seleccionar la red que está utilizando y hacer doble clic en ella.
5. Hacer clic en el botón propiedades, al hacer esto se nos abrirá una ventana con una lista, seleccionamos el apartado Protocolo de Internet versión 4(TCP/IP) y hacemos clic en propiedades.
6. En la ventana que se ha abierto seleccionamos introducir manualmente la dirección de IP, e introducimos la dirección IP que el programa Dynamics asigna para Visual, la mascara de subred se introducirá automáticamente tras introducir la IP y hacer clic en su campo correspondiente (IP por defecto para Visual 192.128.134.184), (en la figura C.3 podemos ver las ventanas de configuración de red).
7. Tras los pasos anteriores, hacemos clic en el botón aceptar y ya estará la configuración de red completada.

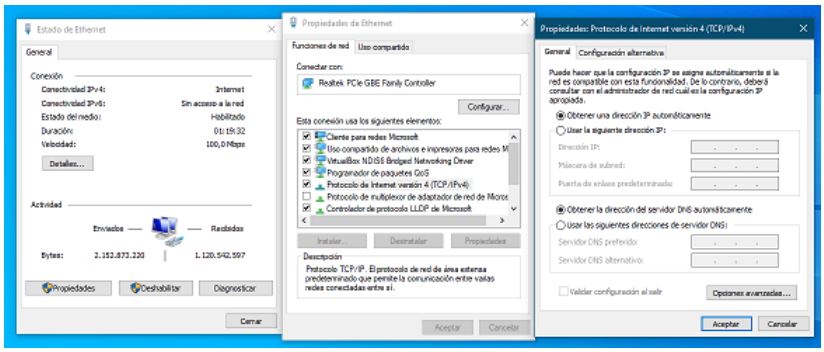

Figura C.3: Sucesión de ventanas de configuración de red.

Tras esto, debemos comprobar que los demás equipos que conforman el simulador tienes las direcciones de IP correctas y que coinciden con las direcciones del fichero host del equipo donde reside Dynamics, tras la comprobación VisualPiperSeneca quedará instalado correctamente.

Bibliografía

[1] Ray L. Page. *Brief History of Flight Simulation*. 2000. url : http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.132.5428&rep=rep1&type=pdf.

[2] Sebastián Rubén Gómez Palomo y Eduardo Antonio Moraleda Gil. *Aproximación a la Ingeniería del Software*. 2014. url : https://books.google.es/books?hl=es&lr=&id=8wnUDwAAQBAJ&oi=fnd&pg=PA19#v=onepage&q&f=false.

[3] Luis Rodriguez De León. *Planificación estratégica, diagrama de Gantt*. 2014. url : https://www.enp.edu.uy/images/libros/Diagrama%20de%20Gantt.pdf.

[4] Mario G Piatinni José A. Calvo Manzano Joaquin Cervera Luis Fernandez. *Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión*. 1996. url : https://bibcatalogo.uca.es/cgi-bin/koha/opac-detail.pl?biblionumber=641213.

[5] Francisco José García Peñalvo y Alicia García Holgado. *Tema 4: Ingeniería de Requisitos*. 2021. url : https://repositorio.grial.eu/bitstream/grial/1143/1/IS_I%20Tema%204%20-%20Ingenieria%20de%20Requisitos.pdf.

[6] Ian Sommerville. *Ingeniería del software*. 2005. url : http://zeus.inf.ucv.cl/~bcrawford/AULA_ICI_3242/Ingenieria%20del%20Software%207ma.%20Ed.%20-%20Ian%20Sommerville.pdf.

[7] Miguel Vega. *Caso de uso*. 2010. url : https://lsi2.ugr.es/~mvega/docis/casos%20de%20uso.pdf.

[8] Roger S.Pressman. *Ingeniería del software un enfoque práctico*. 2010. url : http://www.javier8a.com/itc/bd1/ld-Ingenieria.de.software.enfoque.practico.7ed.Pressman.PDF.

[9] Juan Pavón Mestras. *Patrones de diseño orientados a objetos*. 2004. url : https://www.fdi.ucm.es/profesor/jpavon/poo/2.14PDOO.pdf.

[10] Francisco José García Peñalvo. *Patrones*. 1998. url : https://gredos.usal.es/handle/10366/121937.

[11] Francisco Moreno. *Introducción a la OOP*. 2000. url : https://kataix.umag.cl/~ruribe/Utilidades/Introduccion%20a%20la%20Programacion%20Orientada%20a%20Objetos.pdf.

[12] Stewart Robinson y Gilbert Arbez. *Conceptual modeling : Definition, purpose and benefits*. 2015. url : https://ieeexplore.ieee.org/document/7408386.

[13] Grady Booch Jim Rumbaugh e Ivar Jacobson. *El Lenguaje Unificado de Modelado*. 2004. url : http://elvex.ugr.es/decsai/JAVA/pdf/3E-UML.pdf.

[14] Real academia española. *Definición de protocolo*. 2021. url : https://dle.rae.es/protocolo#otras.

[15] Glenford Myers. *The art of software testing*. 2010. url : https://tinyurl.com/28wkk8bb.

[16] Epic Games. *Documentación de Unreal Engine 4*. 2014. url : `https://docs.unrealengine.com/4.26/en-US/`.

[17] Miguel Angel Quintana Suárez y Miguel Fecanin Araujo. *Introducción a la programación en red: Sockets y Windows Sockets*. 1996. url : `https://tinyurl.com/c834y65w`.